TRABAJO PRÁCTICO N°1
Arquitectura del Software

2C2022

NOMBRE	***PADRÓN***	***MAIL***
Pedro Andrés Flynn	105742	pflynn@fi.uba.ar
Juan Cruz Caserío	104927	jcaserio@fi.uba.ar
Franco Papa	106249	fpapa@fi.uba.ar
Luciano Jadur	101284	ljadur@fi.uba.ar

INTRODUCCIÓN 2

PING 3

PING - Stress 3

Stress - Un Nodo 3

Stress - Multi Nodo 4

PING - Spike 5

Spike - Un Nodo 5

Spike - Multi Nodo 5

PING - Endurance 7

Endurance - Un Nodo 7

Endurance - Multi Nodo 7

SYNC 10

SYNC - Stress 10

Stress - Un Nodo 10

Stress - Multi Nodo 11

SYNC - Spike 12

Spike - Un Nodo 12

Spike - Multi Nodo 12

SYNC - Endurance 14

Endurance - Un Nodo 14

Endurance- Multi Nodo 14

ASYNC

ASYNC - Stress 16

Stress - Un Nodo 16

Stress - Multi Nodo 16

ASYNC - Spike 18

Spike - Un Nodo 18

Spike - Multi Nodo 19

ASYNC - Endurance 20

Endurance - Un Nodo 20

Endurance - Multi Nodo 20

HEAVY 21

HEAVY - Stress 21

Stress - Un Nodo 21

Stress - Multi Nodo 21

HEAVY - Spike 23

Spike - Un Nodo 23

Spike - Multi Nodo 23

HEAVY - Endurance 25

Endurance - Un Nodo 25

Endurance- Multi Nodo 25

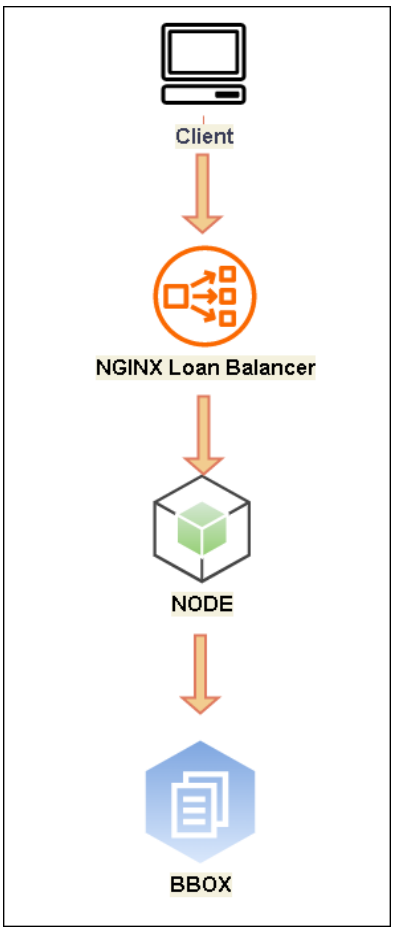
Sección 2
Análisis y caracterización 26

Sección 3
Sistema de Inscripción 29

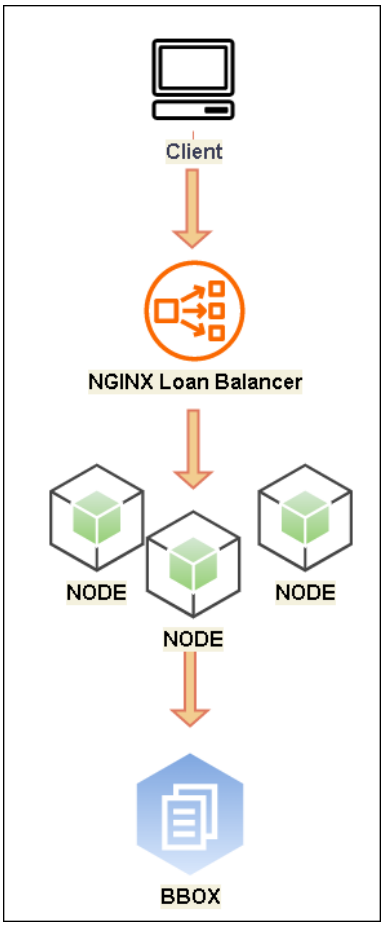
INTRODUCCIÓN

Components & Connectors

Single Node



Multi Node



Tipos de pruebas a realizar:

Sección 1

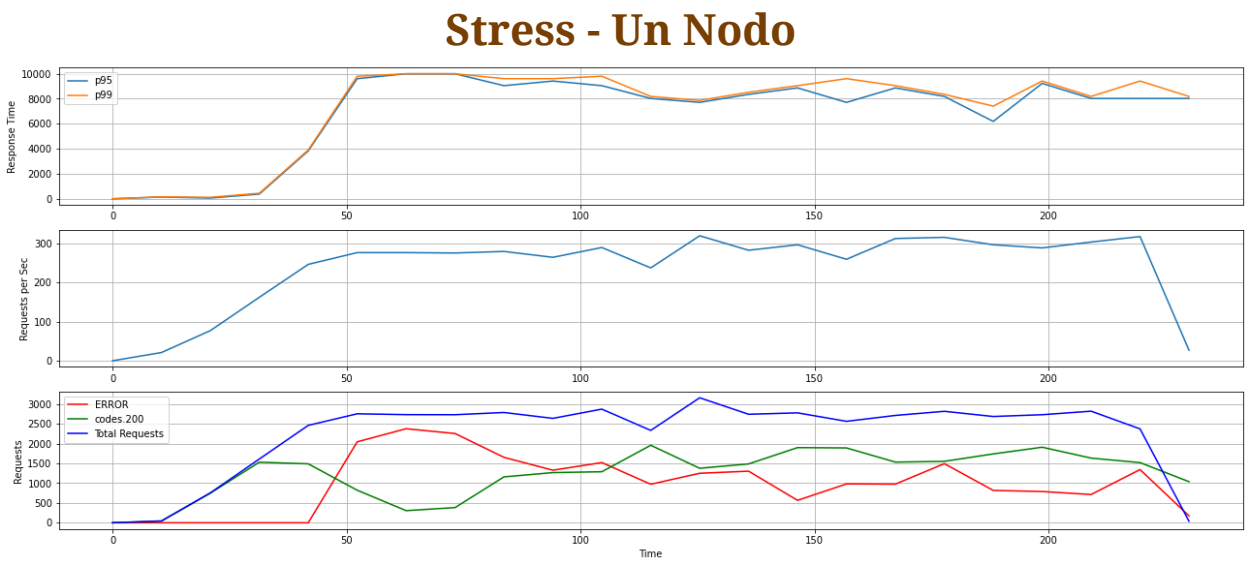
Para la resolución de esta sección pasaremos a listar los diferentes endpoints realizados, y en cada uno de ellos se mostrará como se crearon los escenarios de prueba y cuáles fueron los resultados obtenidos, realizando un análisis del mismo.

PING

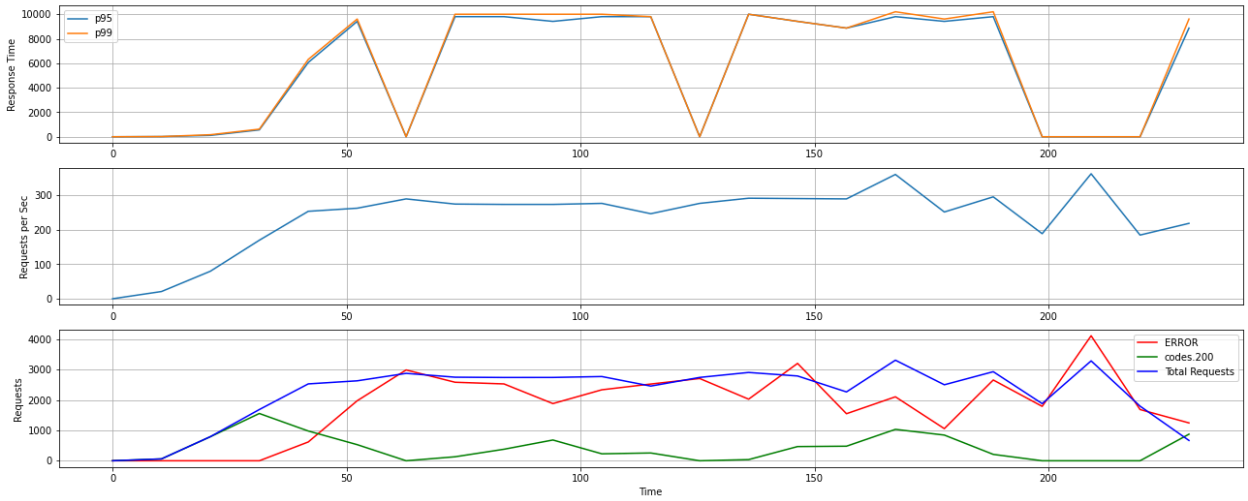
El uso de este endpoint consiste en devolver como respuesta la cadena “Pong” ante cada request recibido. La respuesta no implica ninguna comunicación con servicios externos una vez recibido el request, por lo que su procesamiento es mínimo.

PING - Stress

```
phases:
  - name: Subida
    duration: 30
    arrivalRate: 1
    rampTo: 275
  - name: Constante
    duration: 170
    arrivalRate: 275
  - name: Reset
    arrivalRate: 1
    duration: 20
```



Stress - Multi Nodo



El stress test envía un constante número de requests por segundo en un período prolongado de tiempo (170s).
Es notable que en la fase de “Subida” el endpoint logra responder todos los requests correctamente. Cuando entra a la fase “Constante”, superando la barrera de los 250 RPS, comienza a tener errores de TIMEOUT y de CONNRESET. A lo largo del test, fluctúa entre respuestas positivas y negativas.

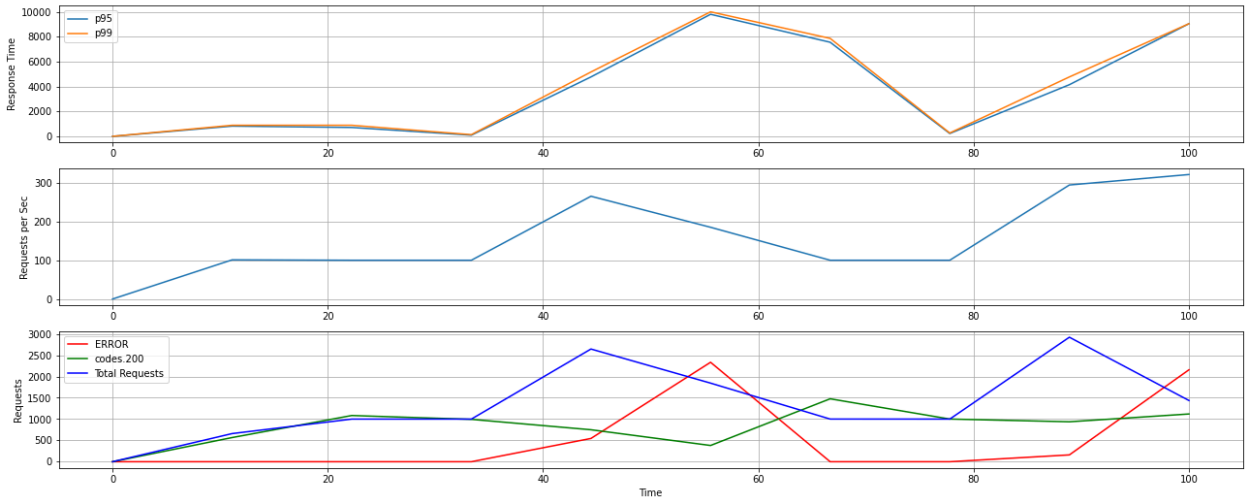
Para el caso de una configuración de varios nodos, el crecimiento y posterior volumen constante de RPS evolucionan de forma muy similar al caso de un único nodo, pero resulta menos eficaz puesto que la tasa de errores obtenidos como respuesta es mayor a lo largo de la fase constante.

La latencia, aunque con algunas fluctuaciones para la configuración multi nodo, es similar en ambos casos, presentando una diferencia de aproximadamente 9 segundos entre la mayor parte de la fase de subida y la fase de carga constante.

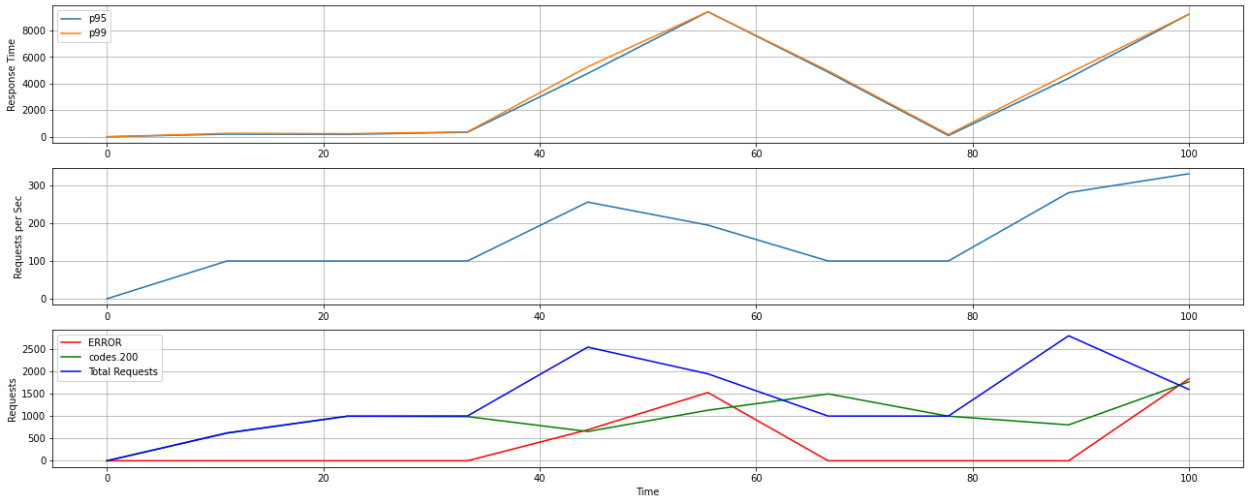
PING - Spike

```yaml
phases:
 - name: Constante
   duration: 30
   arrivalRate: 100
 - name: Pico
   duration: 10
   arrivalRate: 350
 - name: Constante
   duration: 30
   arrivalRate: 100
 - name: Pico
   duration: 10
   arrivalRate: 400
 - name: Reset
   arrivalRate: 1
   duration: 20
```

Spike - Un Nodo



Spike - Multi Nodo



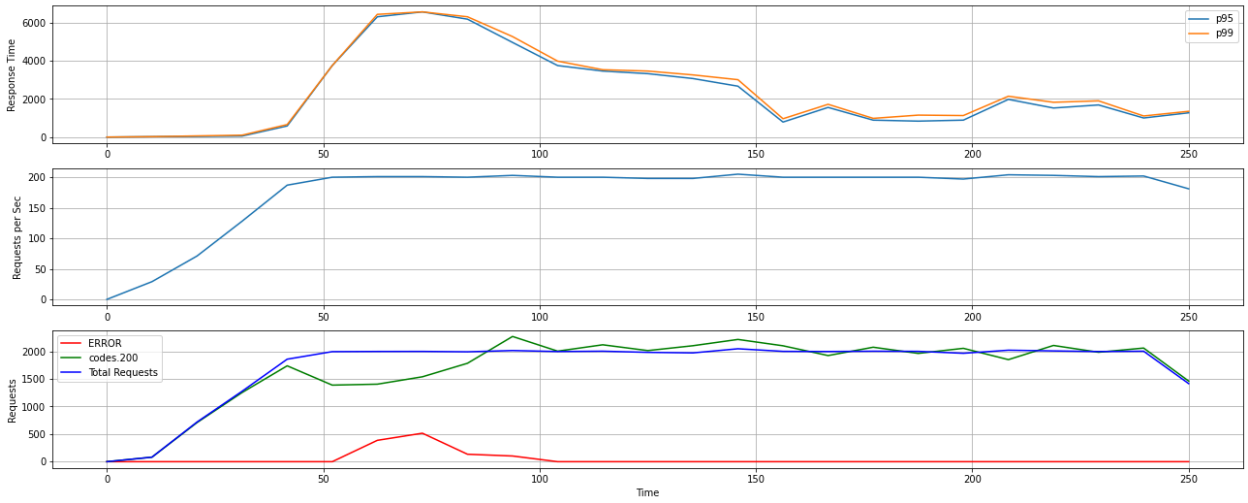
De este test se puede observar que, por un lado, el sistema responde correctamente hasta niveles de carga cercanos a los 100 rps, y comienzan a surgir algunos errores al iniciar el decrecimiento de los volúmenes de rps más altos. Por otro lado, la latencia escala con los incrementos de rps en cada pico.

Con multi nodo se comporta de manera similar, pero pudiendo reducir un poco las respuestas de error.

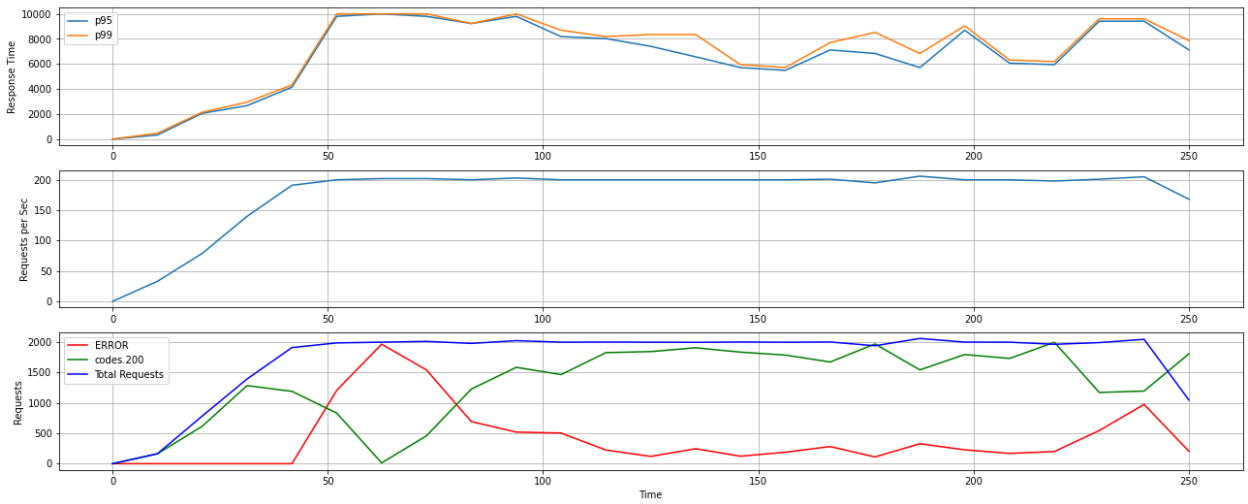
PING - Endurance

```
phases:
 - name: Subida
   duration: 30
   arrivalRate: 20
   rampTo: 200
 - name: Constante
   duration: 200
   arrivalRate: 200
 - name: Reset
   arrivalRate: 1
   duration: 20
```

Endurance - Un Nodo



Endurance - Multi Nodo



Esta prueba presenta una muy buena respuesta en el caso de haber un solo nodo (tope

de RT: 6000 ms) mientras que empeora en multi nodo (tope de RT: 10000 ms). Si bien la diferencia no es muy amplia, la configuración de un único nodo presenta una mejora gradual en la latencia con el paso del tiempo, mientras que los tiempos de respuesta de la configuración multi nodo se mantienen fluctuantes entre 10000 y 6000 ms a lo largo de la fase constante.

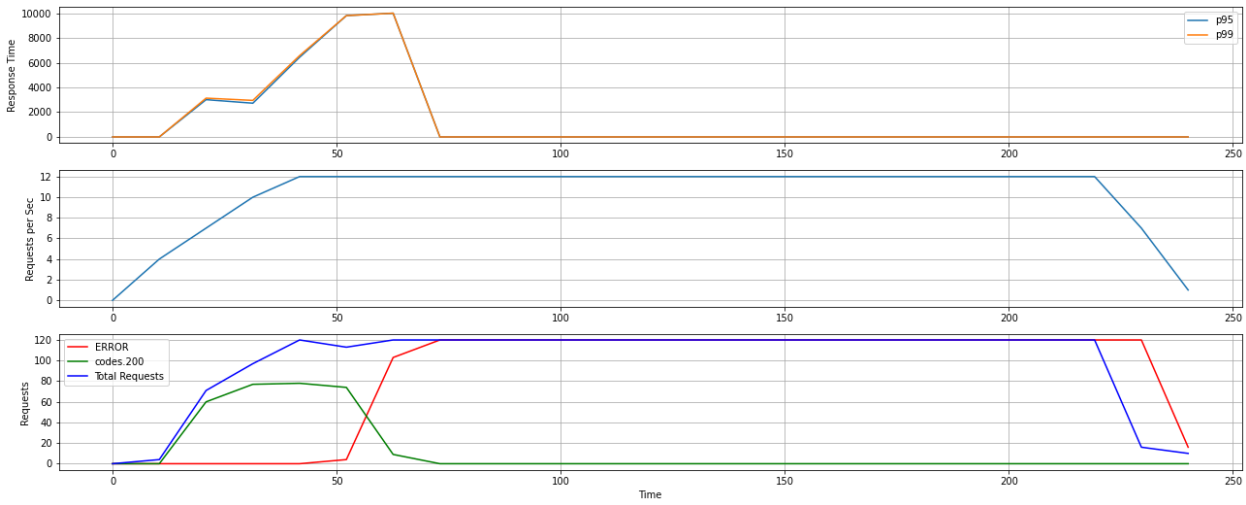
SYNC

El uso de este endpoint consiste en devolver como respuesta la cadena “Sync: Hello world” ante cada request recibido. A diferencia del endpoint **`/async`**, esta cadena es obtenida como respuesta a una petición enviada a **`https://bbox:9091`**.

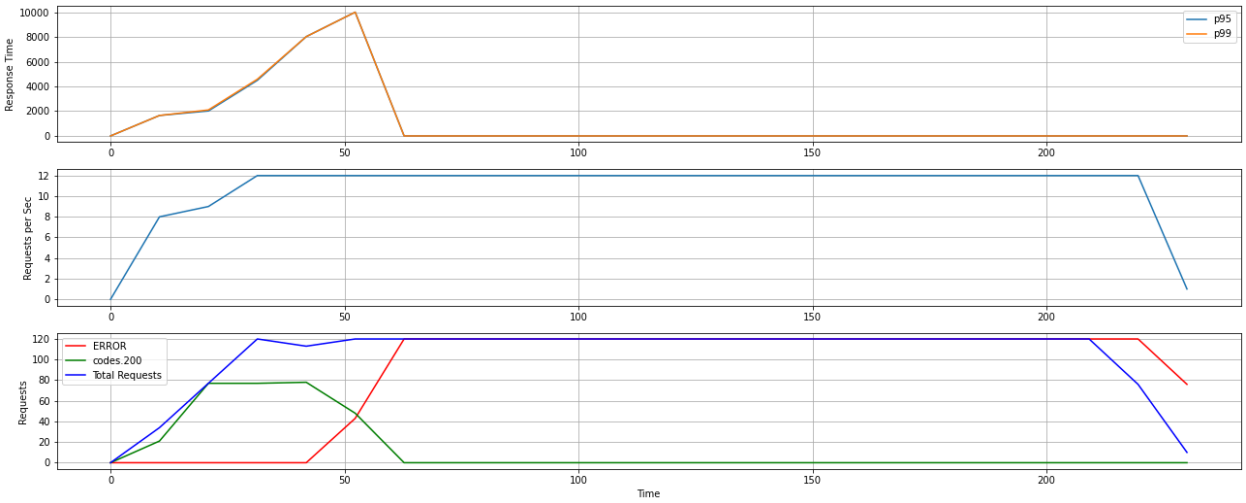
SYNC - Stress

```
phases:
  - name: Subida
    duration: 30
    arrivalRate: 1
    rampTo: 12
  - name: Constante
    duration: 170
    arrivalRate: 12
  - name: Reset
    arrivalRate: 1
    duration: 20
```

Stress - Un Nodo



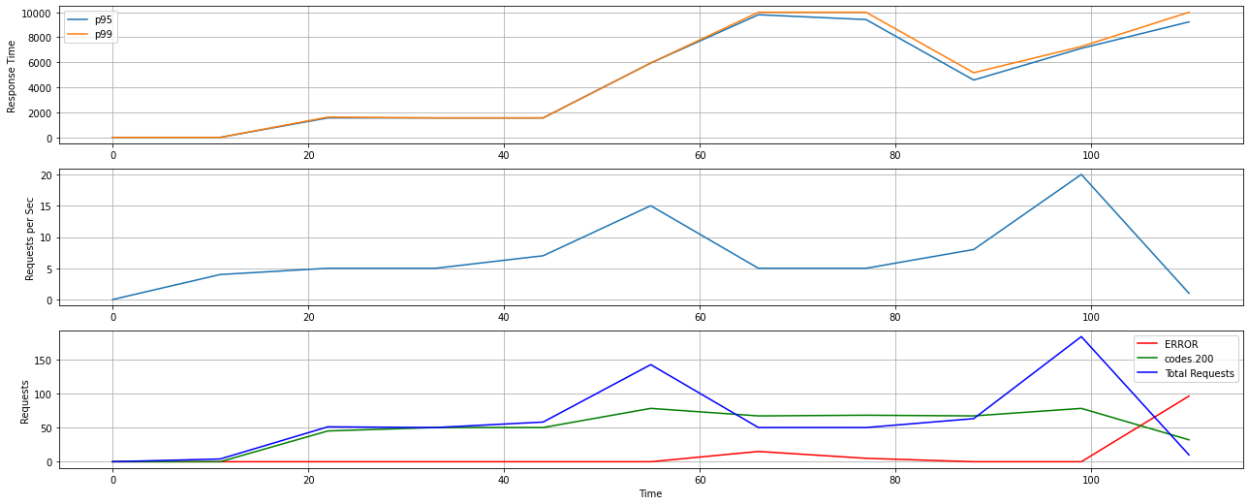
Stress - Multi Nodo



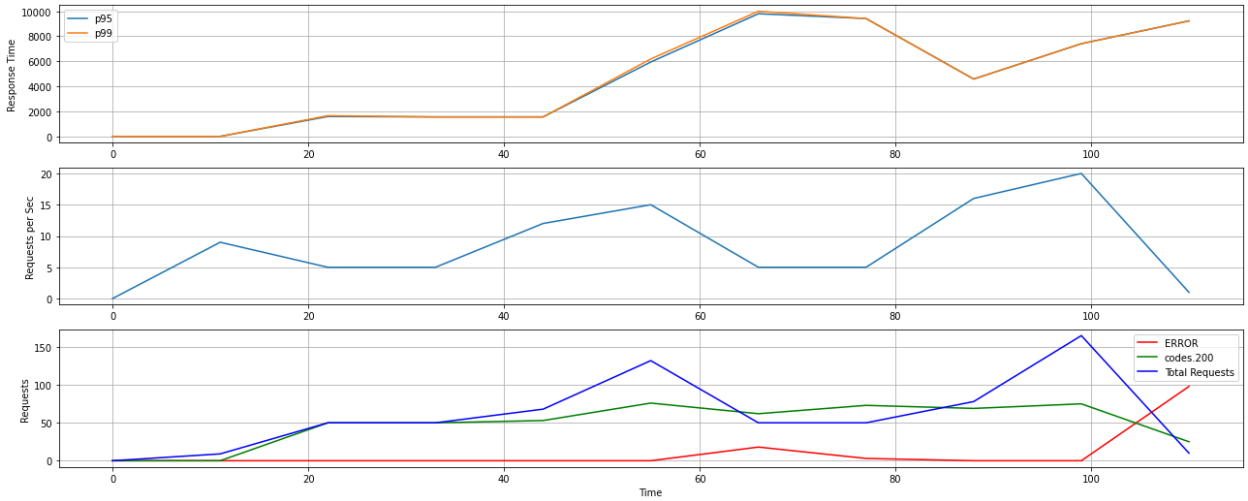
Las pruebas de Stress para este endpoint indican un quiebre del sistema alcanzados los 12 RPS. Por un lado, se puede ver que la latencia crece casi a la par que el incremento de RPS hasta que se alcanza la capacidad máxima de carga, punto a partir del cual el endpoint comienza a responder con error sin capacidad de recuperación hasta que comienza a reducirse la carga recibida.

```yaml
phases:
  - name: Constante
    duration: 30
    arrivalRate: 5
  - name: Pico
    duration: 10
    arrivalRate: 15
  - name: Constante
    duration: 30
    arrivalRate: 5
  - name: Pico
    duration: 10
    arrivalRate: 20
  - name: Reset
    arrivalRate: 1
    duration: 20
```

Spike - Un Nodo



Spike - Multi Nodo

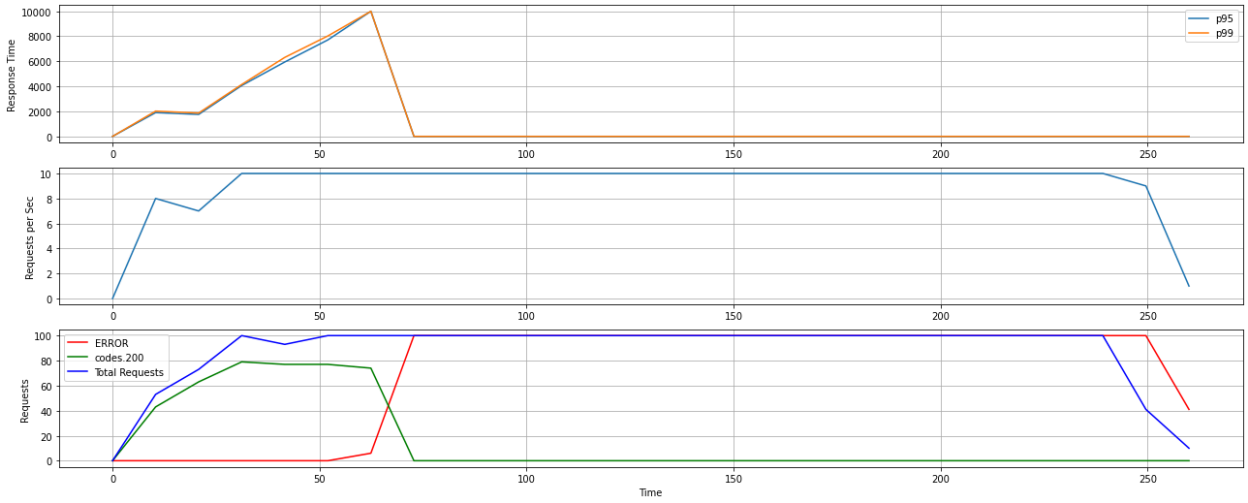


En esta prueba se puede ver que el sistema responde correctamente ante la mayoría de los requests recibidos, pudiendo soportar picos esporádicos de alto incremento en el volumen RPS, y se puede generalizar este comportamiento independientemente de la cantidad de contenedores en los que se encuentre replicado el sistema.

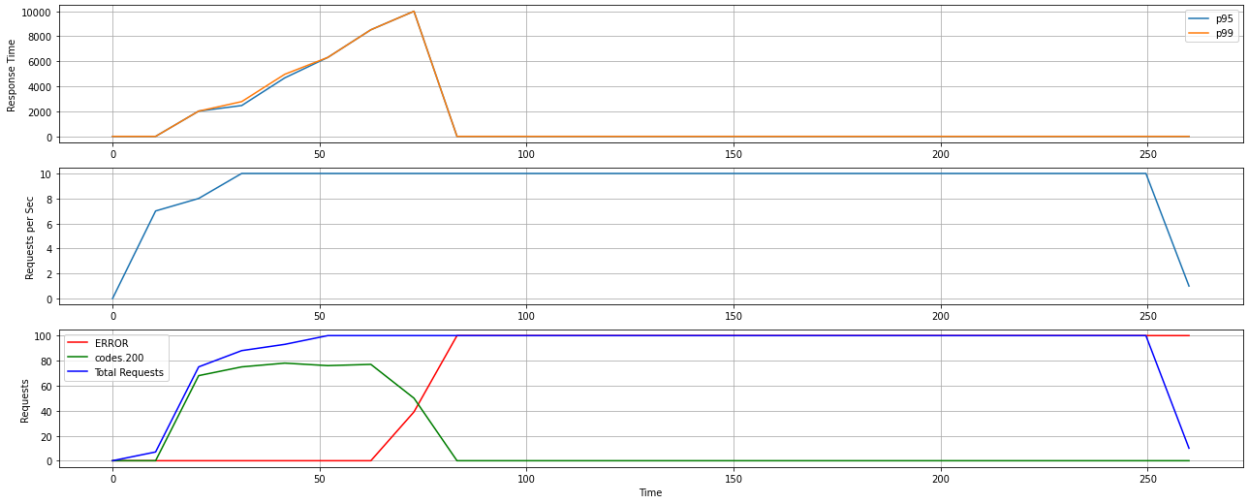
Por otro lado, también es posible apreciar que la latencia aumenta considerablemente ante la aparición de algún pico de altos niveles de carga, y no decrece rápidamente sino que, cómo se puede ver en ambos gráficos, demora alrededor de unos 35 segundos en comenzar a reducir los tiempos de respuesta una vez superado el máximo de RPS obtenido en cada pico, lo que puede implicar problemas de performance no despreciables en escenarios donde un mismo endpoint reciba altos volúmenes de request en pequeños intervalos de tiempo con frecuencias muy altas.

```yaml
phases:
  - name: Subida
    duration: 30
    arrivalRate: 1
    rampTo: 10
  - name: Constante
    duration: 200
    arrivalRate: 10
  - name: Reset
    arrivalRate: 1
    duration: 20
```

Endurance - Un Nodo



Endurance- Multi Nodo



En esta prueba se puede ver que el endpoint soporta una carga de unos 10 RPS durante aproximadamente un minuto desde el comienzo de la fase constante, tras lo cual el sistema comienza a fallar y no logra recuperarse.

La latencia crece de forma lineal hasta dicho punto de quiebre, llegando a durar hasta cerca de 10000 ms para 10 RPS, lo que sugiere una no muy buena performance para atender a múltiples usuarios en simultáneo.

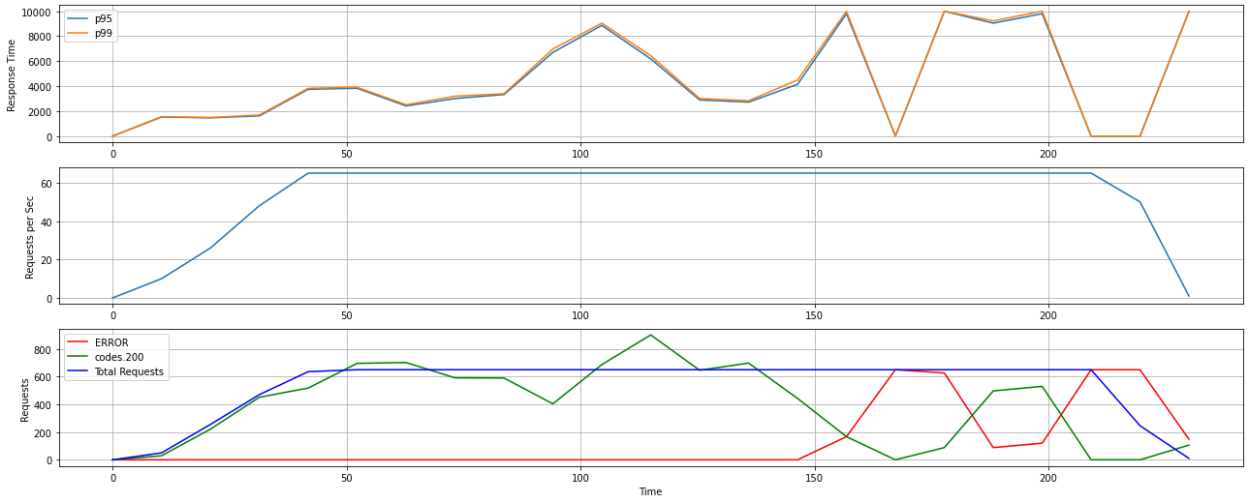
ASYNC

El uso de este endpoint consiste en devolver como respuesta la cadena "*Async: Hello world*" ante cada request recibido. Dicha cadena es obtenida como respuesta a una petición enviada a `https://bbox:9090`. En principio, no sabíamos si era sincrónico o asincrónico. La justificación está en la **Sección 2**.

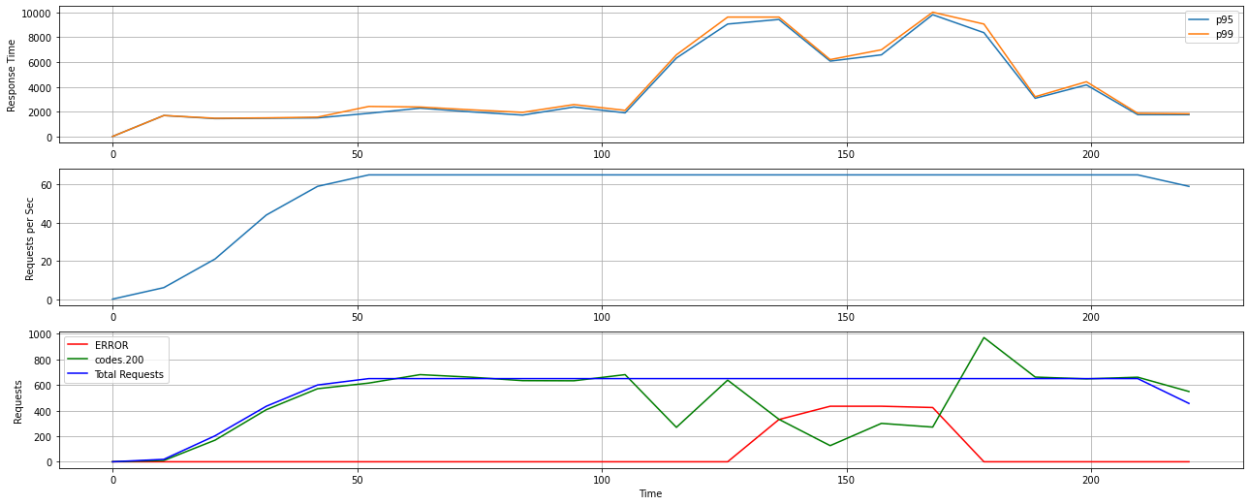
ASYNC - Stress

```yaml
phases:
  - name: Subida
    duration: 30
    arrivalRate: 1
    rampTo: 65
  - name: Constante
    duration: 170
    arrivalRate: 65
  - name: Reset
    arrivalRate: 1
    duration: 20
```

Stress - Un Nodo



Stress - Multi Nodo



Planteando un escenario idéntico al del endpoint /sync, para este caso se obtuvieron resultados diametralmente opuestos. Si bien la latencia encuentra algunas fluctuaciones, las tasas de *error responses* resultaron nulas para ambas configuraciones, lo que señala que, en mismas condiciones de carga, la performance de este endpoint es muy superior a la de su contraparte asincrónica.

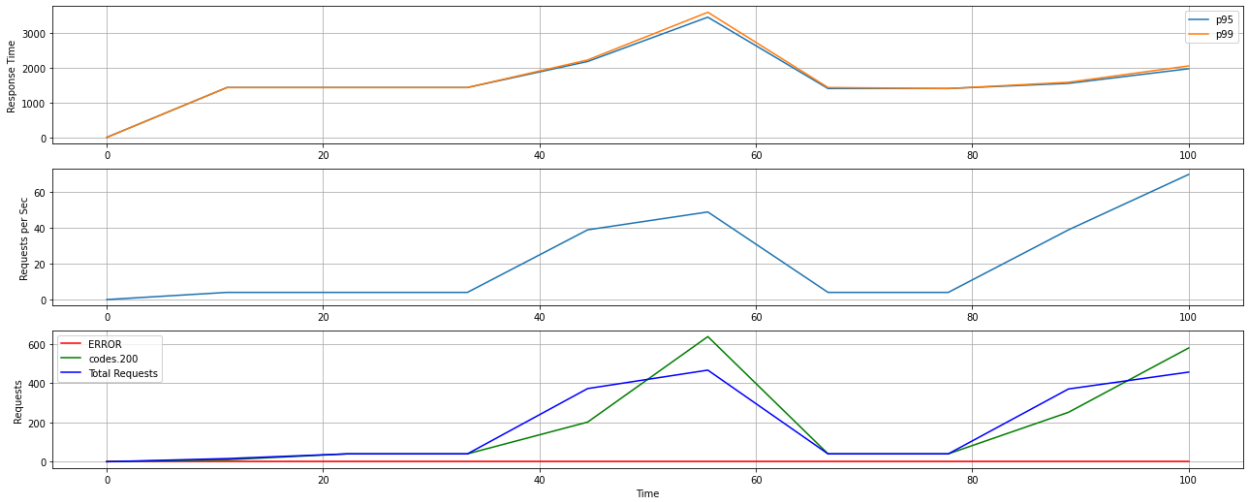
En cuanto a las configuraciones del sistema probadas con este endpoint, la diferencia principal que se puede notar reside en las latencias: aunque ambas presentan algunas fluctuaciones en torno a valores muy similares (medidos en segundos), para la configuración de un único nodo los tiempos de respuesta presentan fluctuaciones con mayor frecuencia y con saltos bruscos cuando comienzan a surgir intervalos temporales donde predominan los errores, mientras que en un sistema de múltiples nodos la latencia, si bien alcanza máximos similares a los de la arquitectura anterior, mantiene niveles más bajos y estables de latencia, y los incrementos sólo ocurren cuando comienza a haber fallas en las respuestas.

Aunque de forma sutil, se puede afirmar que el sistema con múltiples containers presenta una mejor eficiencia que el de un solo nodo al correr este endpoint en escenarios de recursos limitados.

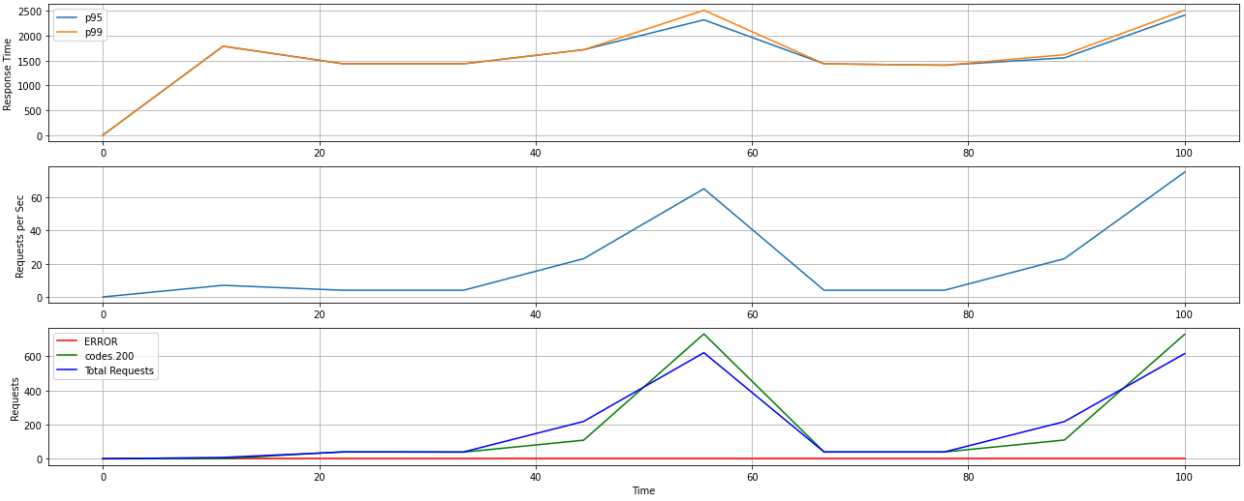
ASYNC - Spike

```yaml
phases:
 - name: Constante
   duration: 30
   arrivalRate: 4
 - name: Pico
   duration: 10
   arrivalRate: 80
 - name: Constante
   duration: 30
   arrivalRate: 4
 - name: Pico
   duration: 10
   arrivalRate: 80
 - name: Reset
   arrivalRate: 1
   duration: 20
```

Spike - Un Nodo



Spike - Multi Nodo



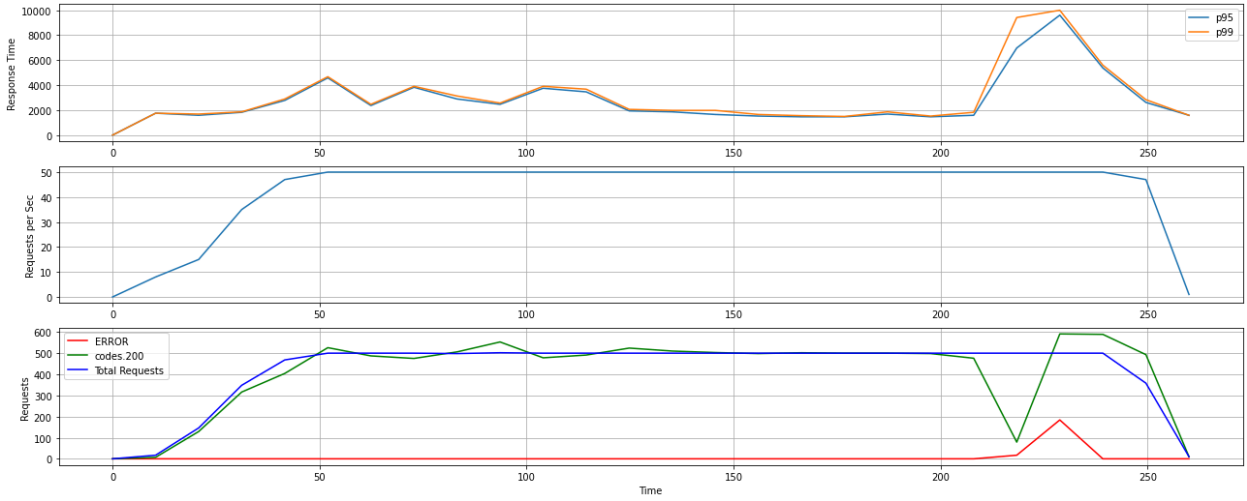
Similar a su contraparte en /sync, el sistema resiste sin problemas la aparición de incrementos pronunciados de carga en pequeños intervalos de tiempo, mostrando una eficacia aún mejor, tal como se puede ver en los resultados obtenidos: el volumen de errores es directamente despreciable.

Los tiempos de respuesta se comportan de forma muy similar en ambas configuraciones puesto que aumentan con la aparición de picos de RPS y se mantienen estables el resto del tiempo; no obstante, ante los crecimientos de latencia el sistema replicado en múltiples contenedores se muestra más eficiente puesto que alcanza picos de latencia de menor valor que su contraparte de único contenedor.

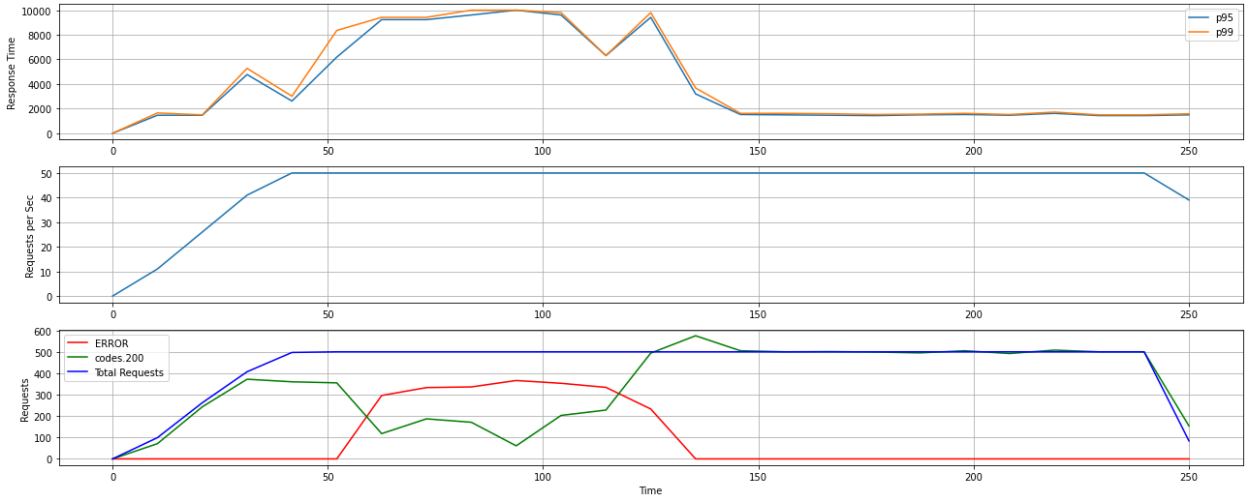
ASYNC - Endurance

```yaml
phases:
 - name: Subida
   duration: 30
   arrivalRate: 1
   rampTo: 50
 - name: Constante
   duration: 200
   arrivalRate: 50
 - name: Reset
   arrivalRate: 1
   duration: 20
```

Endurance - Un Nodo



Endurance - Multi Nodo



Nuevamente, al correr esta prueba se obtuvieron resultados considerablemente favorables en comparación con el endpoint /sync. Dado un alto volumen de RPS sostenido a lo largo del tiempo, el volumen de fallas devueltas por el sistema compuesto por un único contenedor es casi nulo, mientras que para la configuración multi nodo el procesamiento de requests es más propenso a errores al inicio de la fase constante, pero logra recuperarse al cabo de un minuto y medio aproximadamente, para una carga constante de 50 RPS. Además, nuevamente se obtiene una variación de la latencia que escala con el crecimiento del volumen de errores, resultando aún así en una obtención del tiempo de respuesta generalmente más eficiente que en los dos endpoints anteriores.

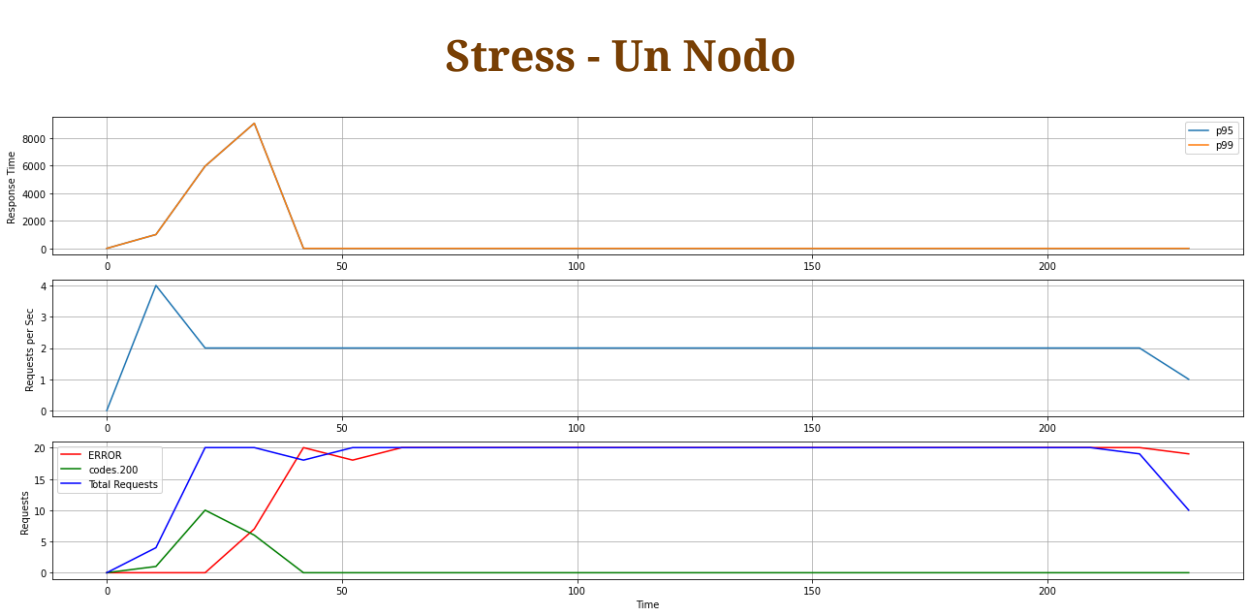
De estos resultados se puede concluir que este endpoint escala mejor en sistemas replicados en un único contenedor.

HEAVY

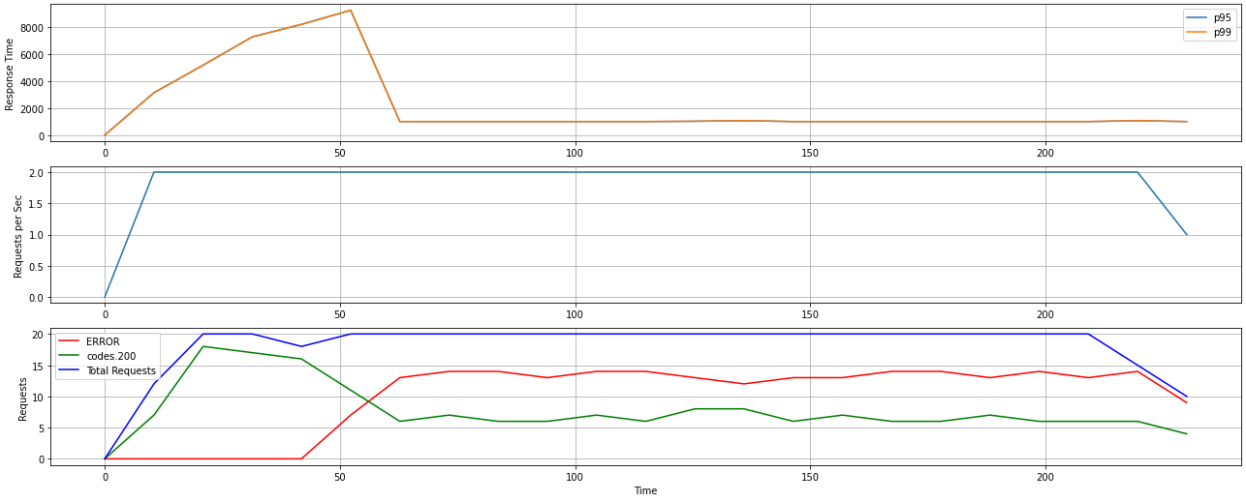
Para esta prueba usaremos un endpoint que se ocupe de generarle un gran esfuerzo al procesador, de manera tal de producir un latencia más grande y seguramente un response time peor ya que deberá ocupar más tiempo por cada request.

HEAVY - Stress

```yaml
phases:
  - name: Subida
    duration: 30
    arrivalRate: 1
    rampTo: 2
  - name: Constante
    duration: 170
    arrivalRate: 2
  - name: Reset
    arrivalRate: 1
    duration: 20
```



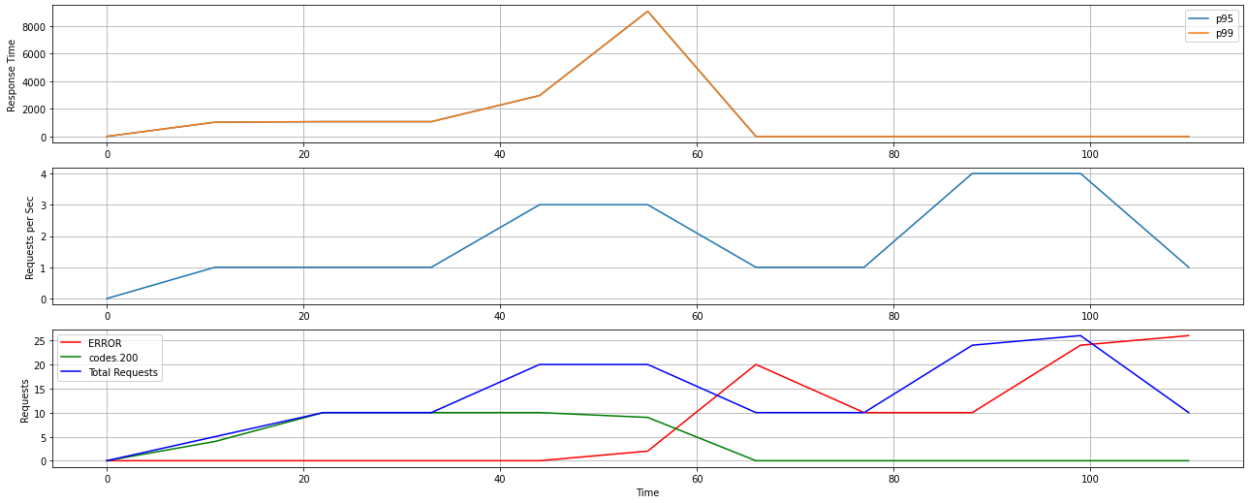
Stress - Multi Nodo



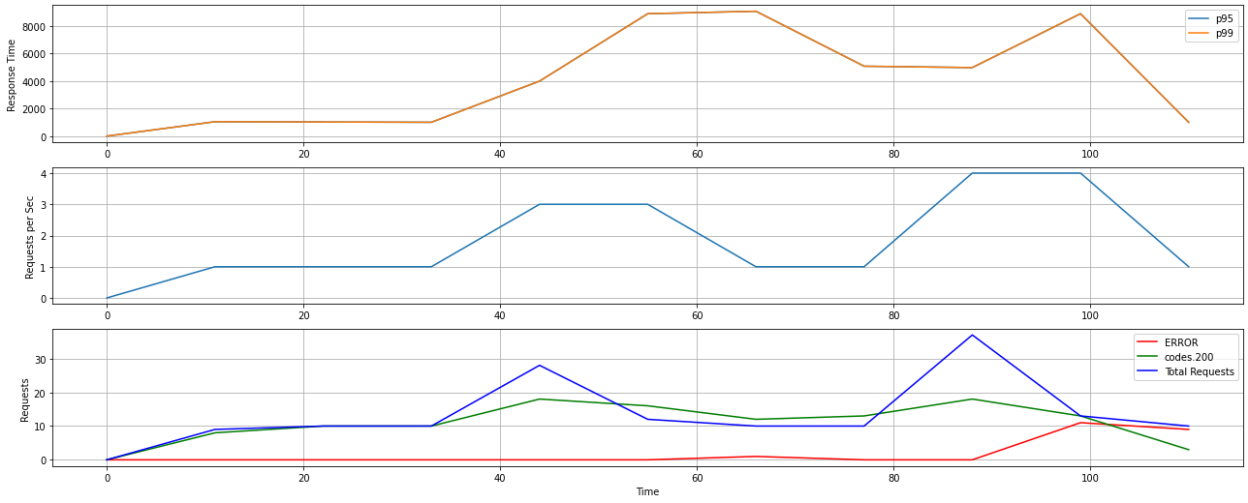
Se puede ver que el sistema no es muy robusto frente a pruebas de estrés, independientemente de la cantidad de contenedores en la que esté replicado. Al cabo de un poco menos de un minuto el volumen de errores retornados se vuelve predominante y no hay señales de recuperación, nada menos que ante una carga sostenida de 2 RPS, lo que señala que el sistema no está diseñado para soportar altas cargas de procesamiento para múltiples usuarios en simultáneo.

```yaml
phases:
  - name: Constante
    duration: 30
    arrivalRate: 1
  - name: Pico
    duration: 10
    arrivalRate: 3
  - name: Constante
    duration: 30
    arrivalRate: 1
  - name: Pico
    duration: 10
    arrivalRate: 4
  - name: Reset
    arrivalRate: 1
    duration: 20
```

Spike - Un Nodo



Spike - Multi Nodo



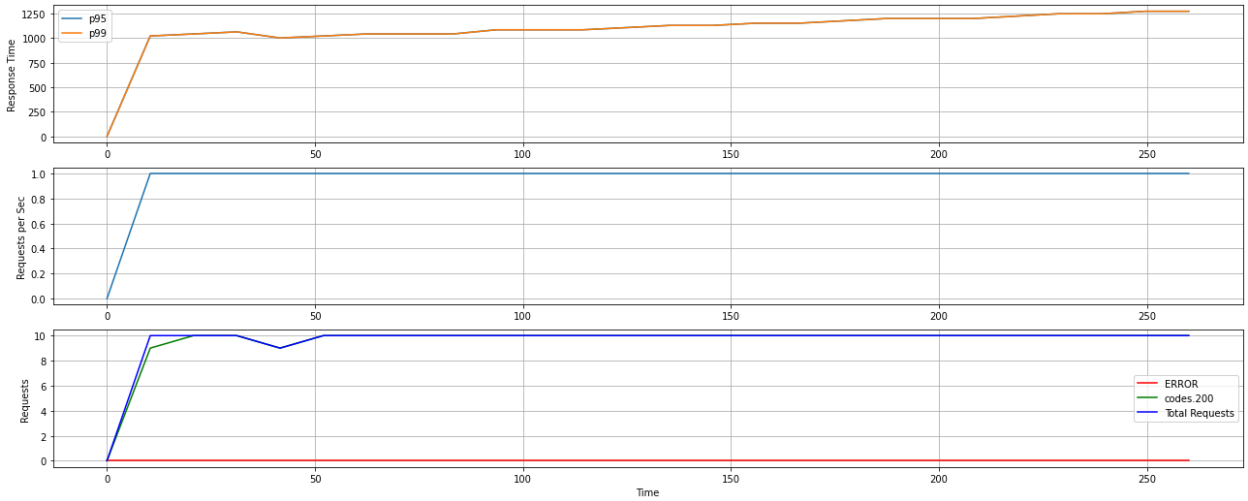
A diferencia que con las pruebas de estrés, el sistema se muestra confiable ante picos esporádicos de carga para configuraciones con múltiples nodos, mientras que con un único nodo la tasa de error vuelve a predominar una vez finalizado el primer pico, sin señales de recuperación a futuro. No obstante, se puede apreciar que, para la configuración multi nodo, tras alcanzar picos cercanos a los 40 RPS el la proporción de errores aumenta hasta volverse predominante, lo cual sugiere la posibilidad de que haya algún límite de carga para incrementos bruscos de request en simultáneo en torno a esa cifra para el cual el sistema deja de ser eficaz.

Por otro lado, en ambos casos -mientras funcione correctamente- se nota un considerable aumento en la latencia del endpoint al alcanzar los extremos de cada pico, algo que a priori es esperable debido a que se supone que se están procesando grandes cantidades de operaciones por cada request recibido.

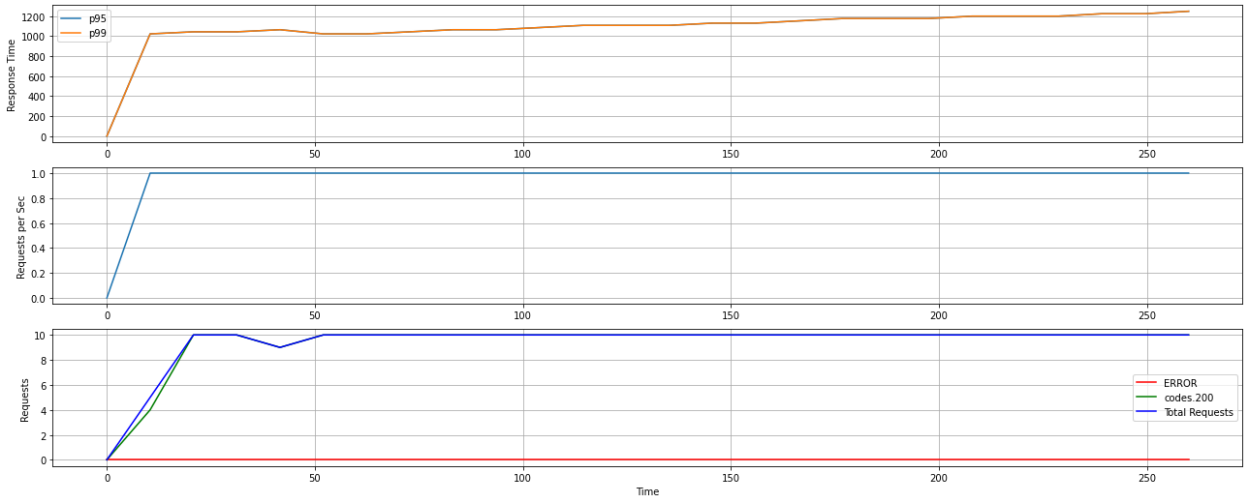
HEAVY - Endurance

```yaml
phases:
  - name: Subida
    duration: 30
    arrivalRate: 1
    rampTo: 1
  - name: Constante
    duration: 170
    arrivalRate: 1
  - name: Reset
    arrivalRate: 1
    duration: 20
```

Endurance - Un Nodo



Endurance- Multi Nodo



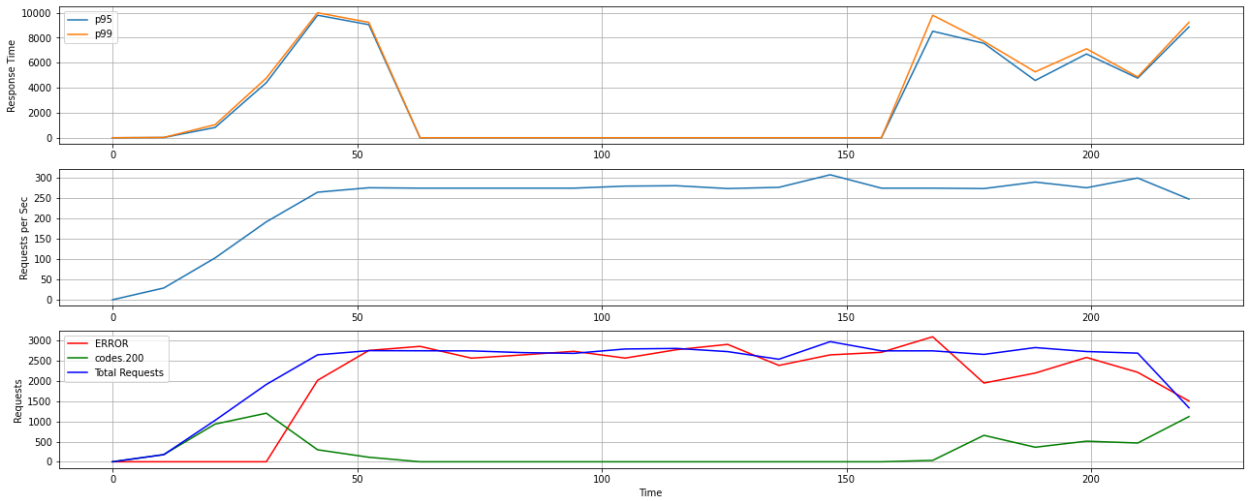
.

Métricas desde la App

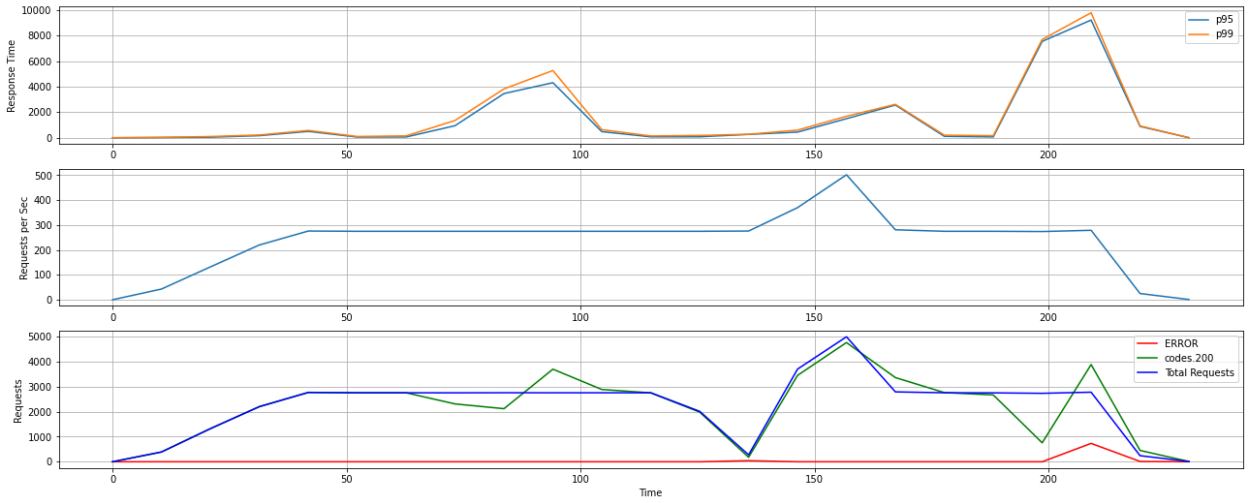
Usando *lynx* desde la aplicación, logramos que *statsd* pueda capturar las métricas desde el punto de vista de esta. Mostraremos los mismos tests para la seccion 1 pero aplicados a esta métrica.

PING

Stress test - Single node

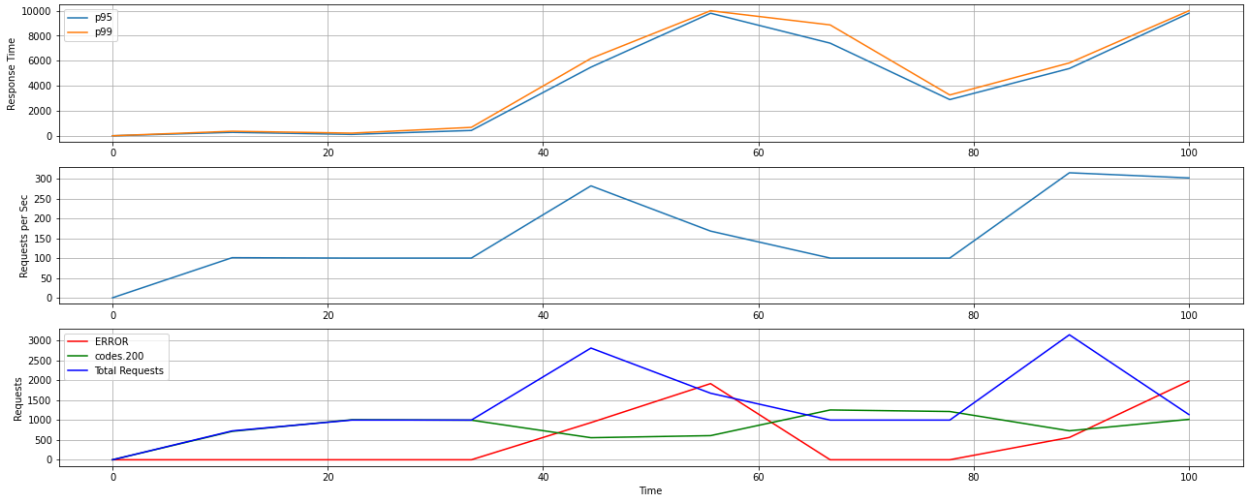


Stress test - Multi node

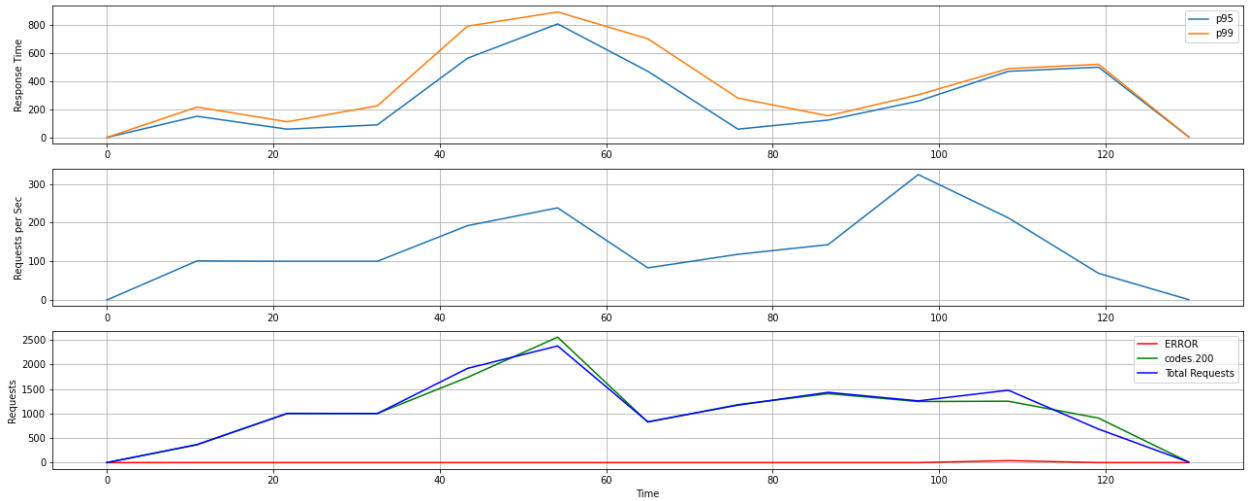


En multi nodo logra llegar a 500 rps manteniendo un promedio de response time de 2000 ms (respondiendo todos con status 200), mientras que en single node la app falla completamente a partir de los 200 rps.

Spike test - Single node

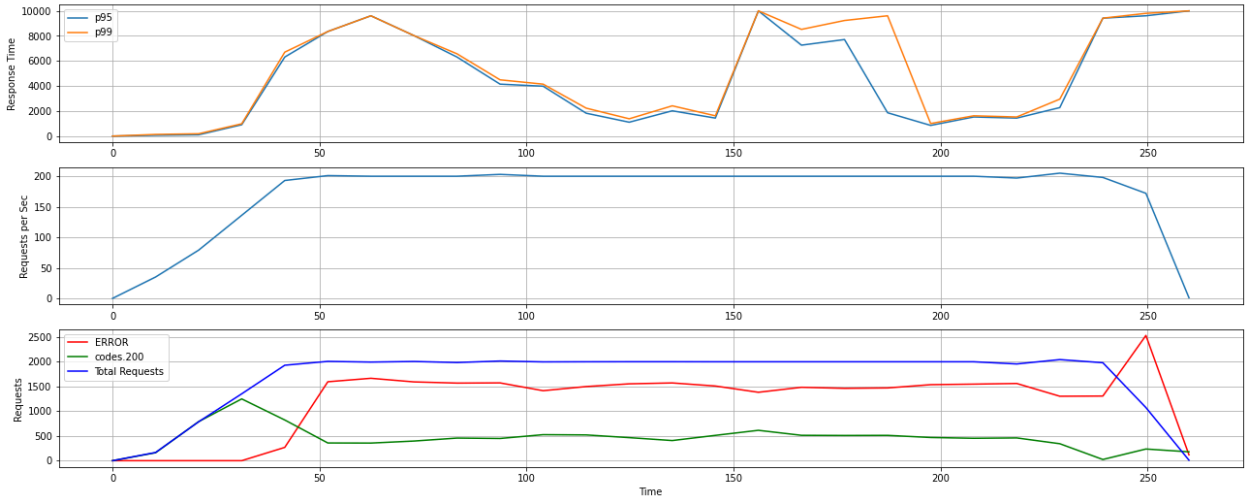


Spike test - Multi node

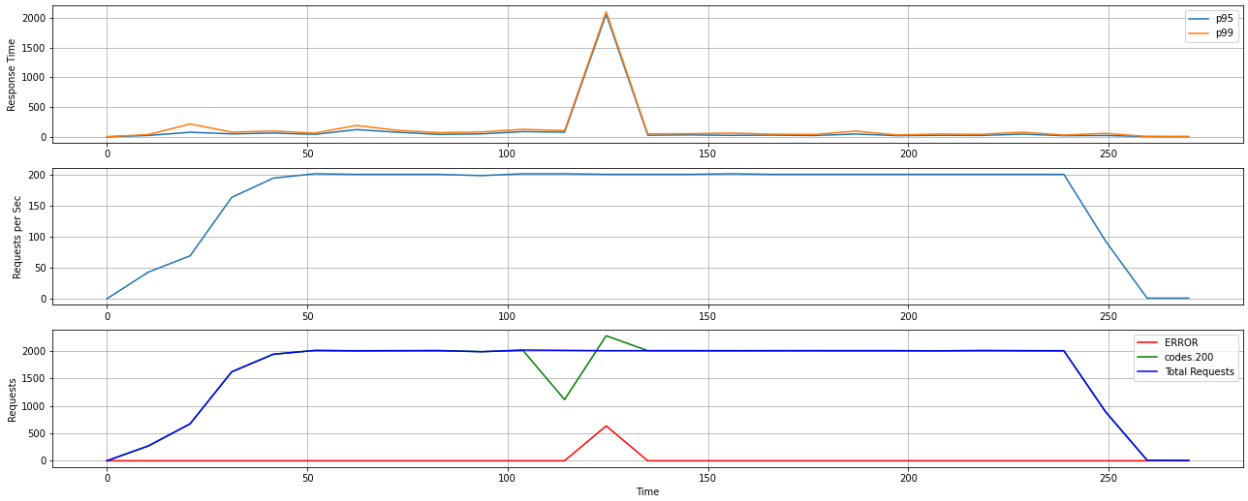


En el spike test la app falla en los puntos críticos del test para single node.
Multi node se sigue manteniendo sin errores con un response time maximo de 800 que comparándolo con las métricas originales da muchísimos mejores resultados.

Endurance test - Single node

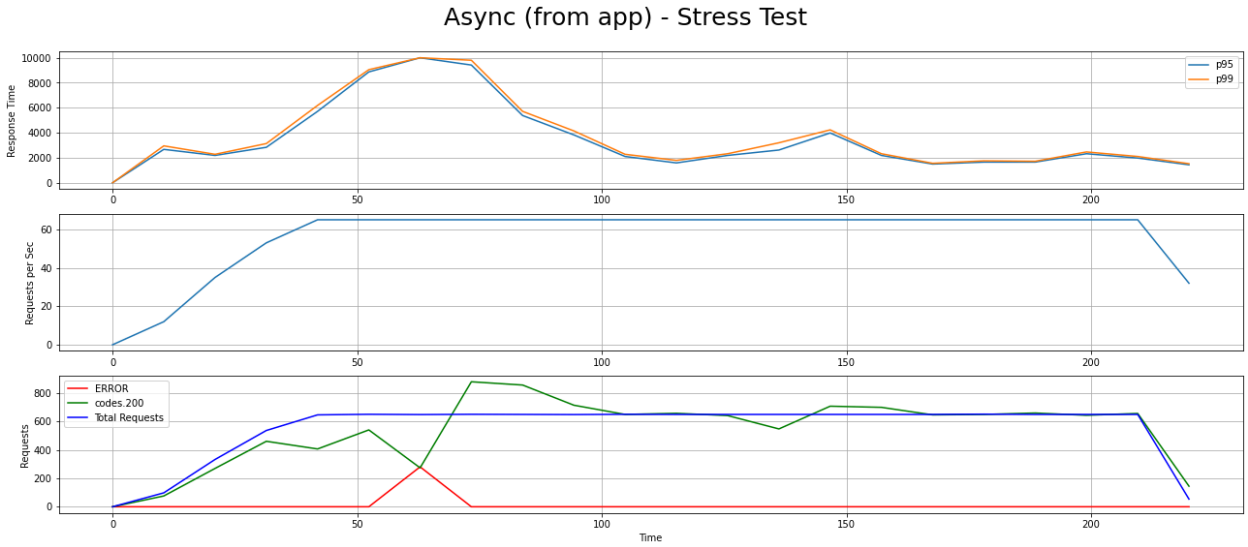


Endurance test - Multi node

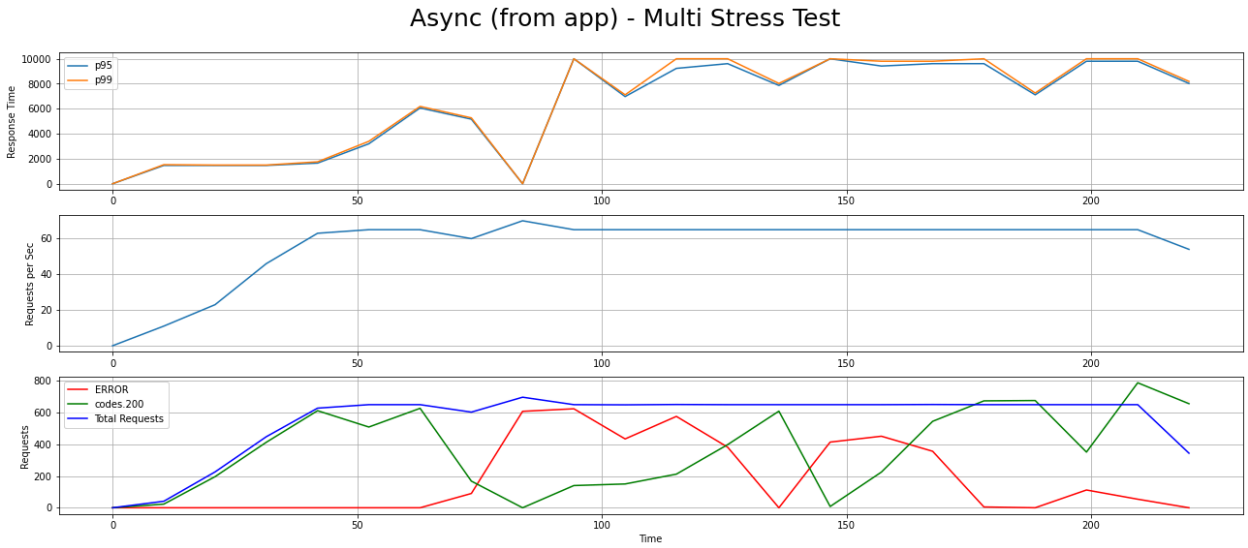


ASYNC

Stress test - Single node

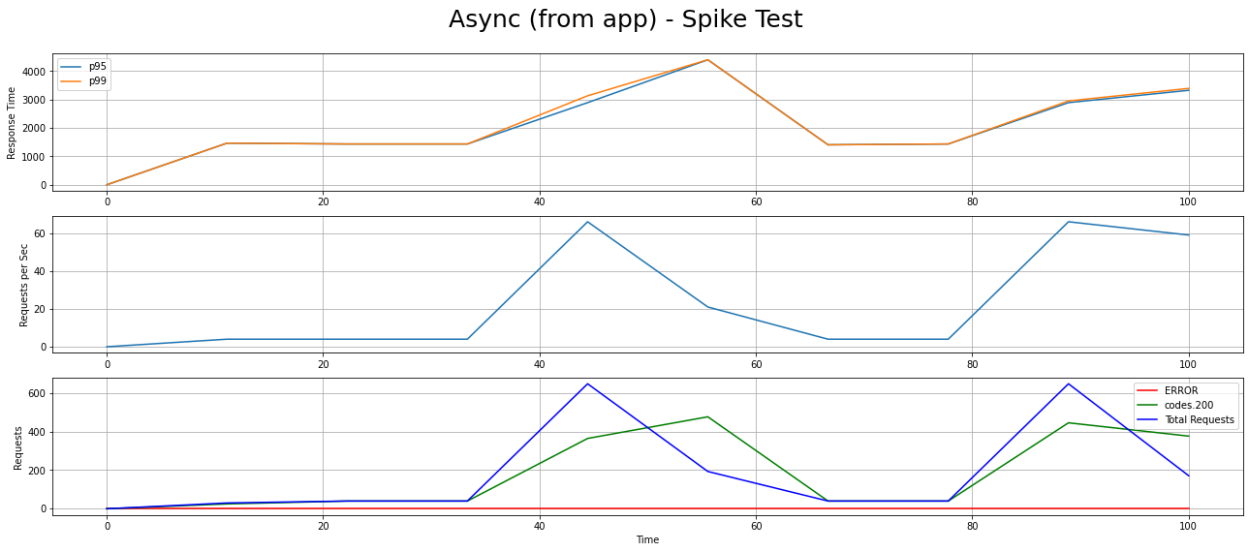


Stress test - Multi node

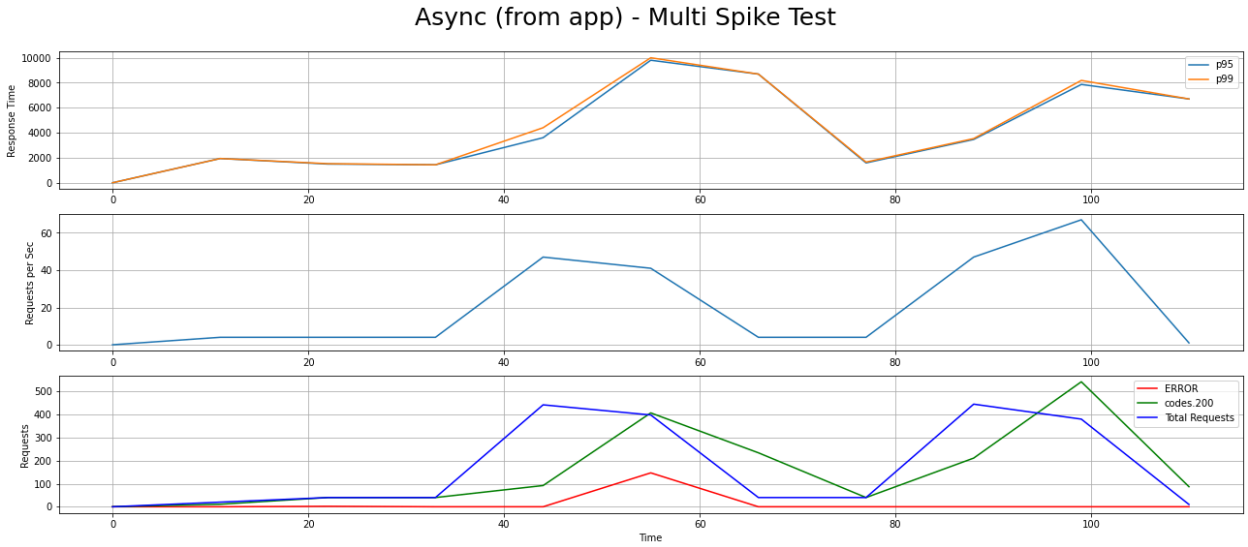


Aquí los resultados son diversos, a simple vista parece que el multi node es peor que el single.

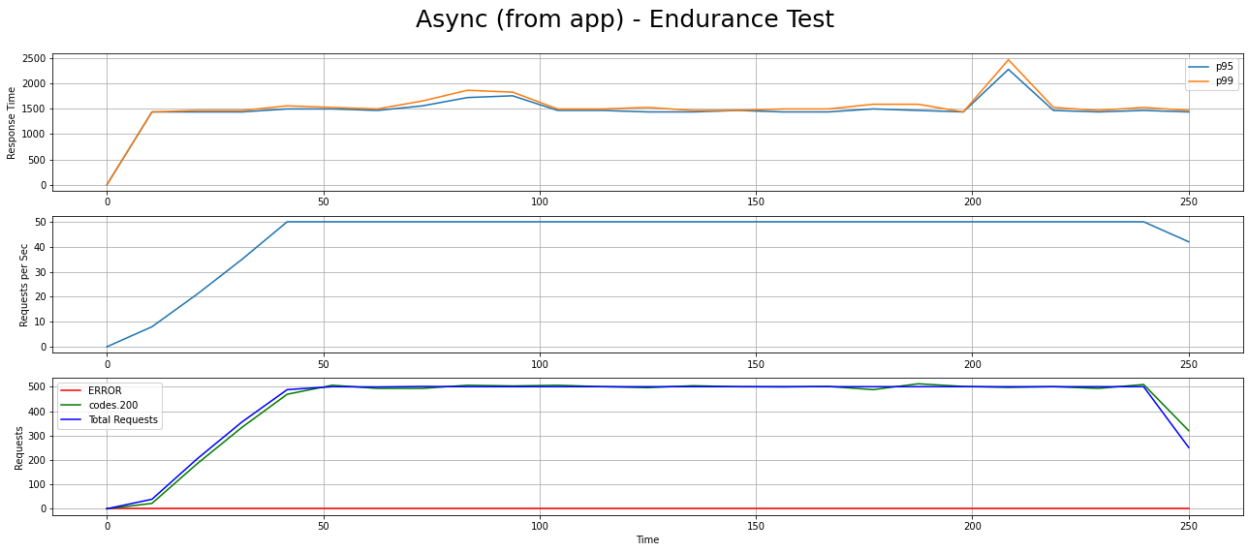
Spike test - Single node



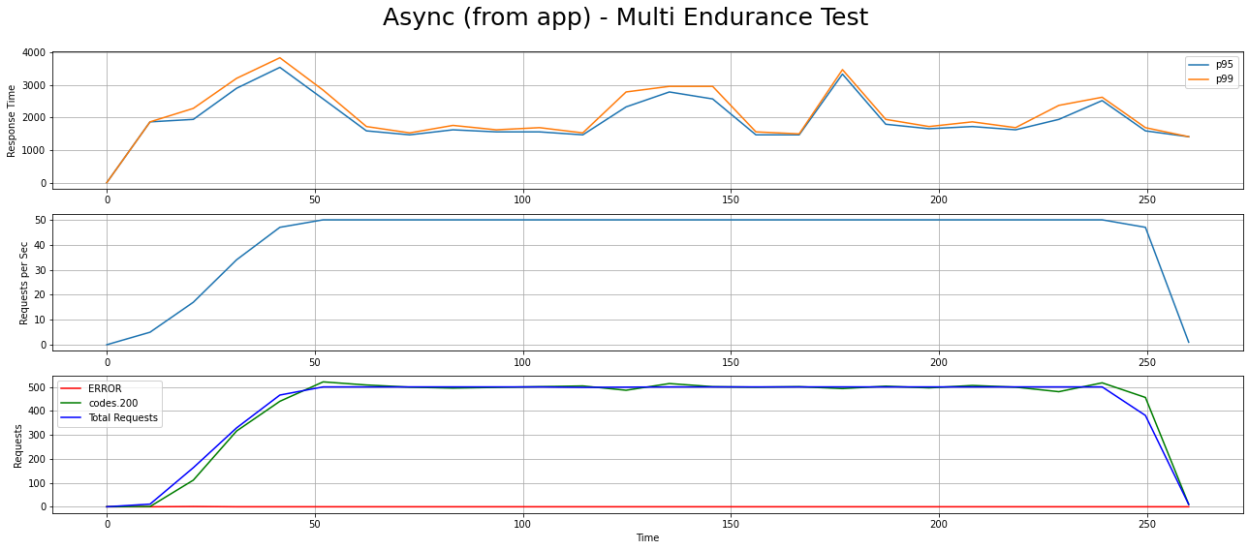
Spike test - Multi node



Endurance test - Single node



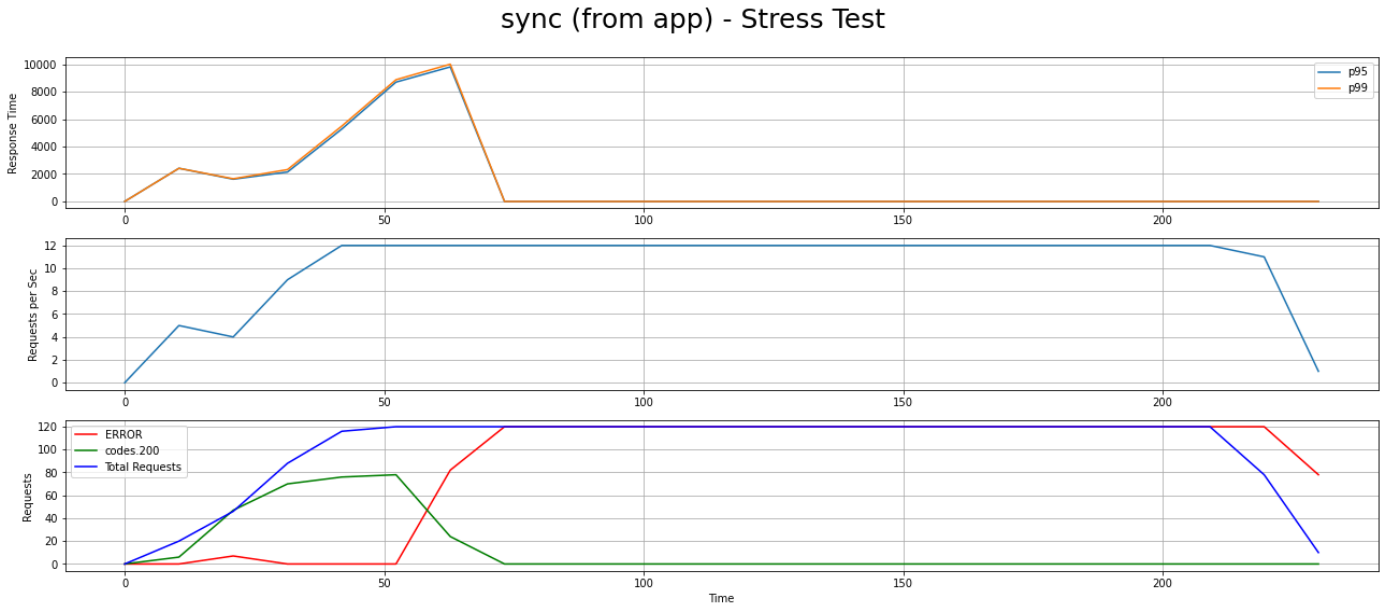
Endurance test - Multi node



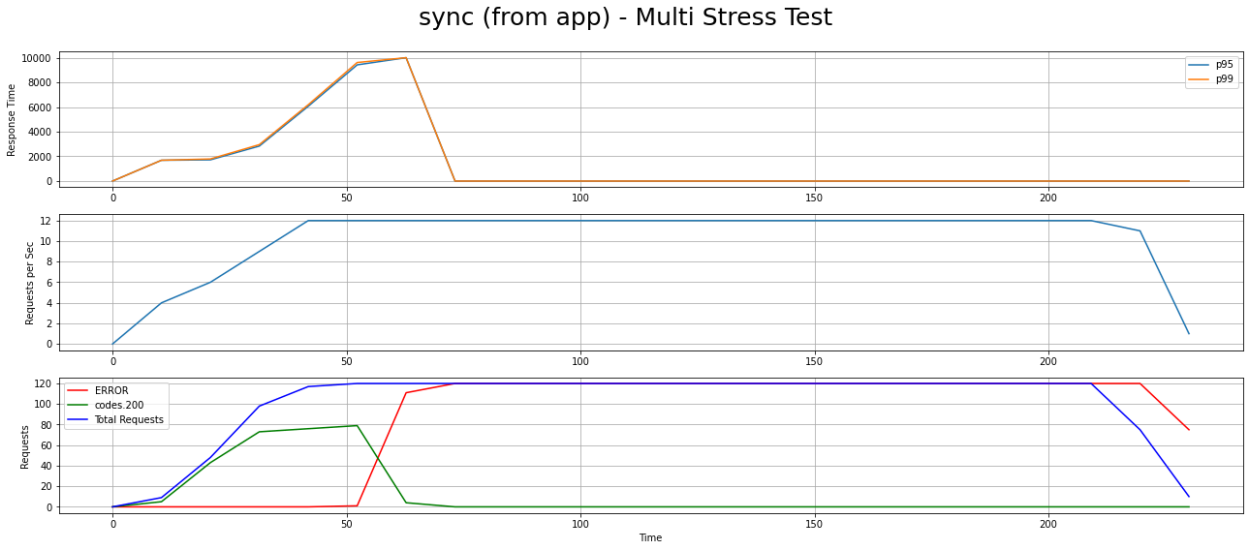
Aquí los resultados son diversos, a simple vista parece que el multi node es peor que el single.

SYNC

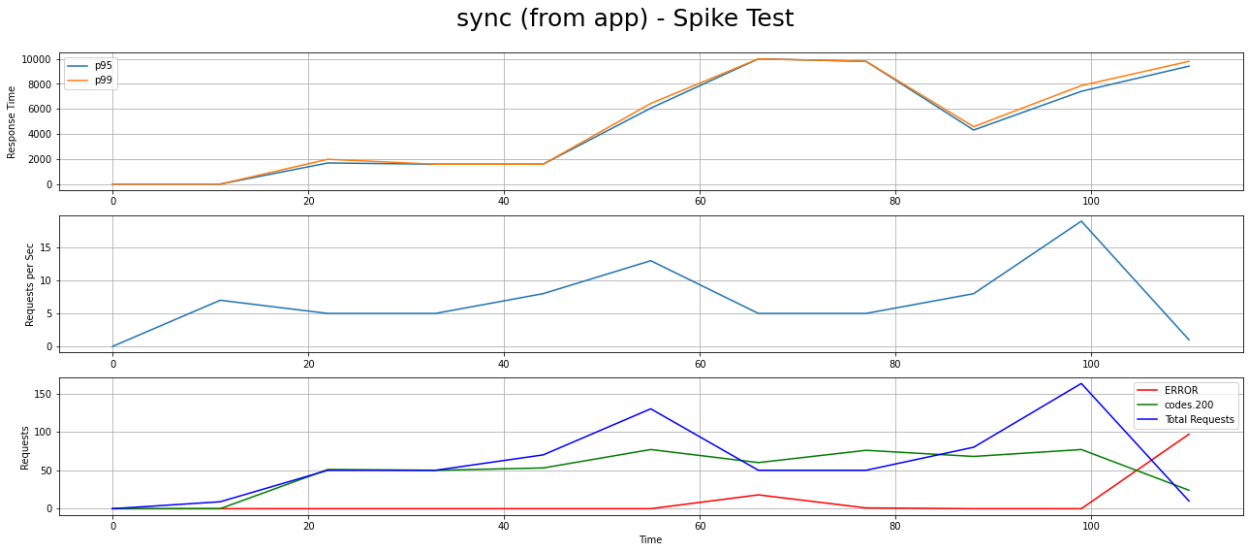
Stress test - Single node



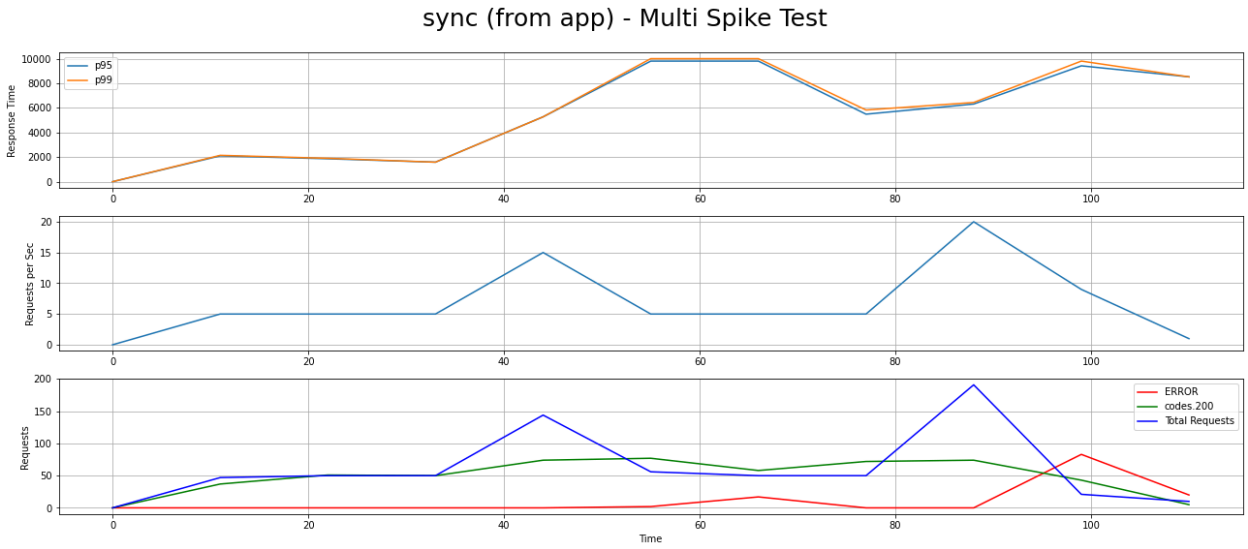
Stress test - Multi node



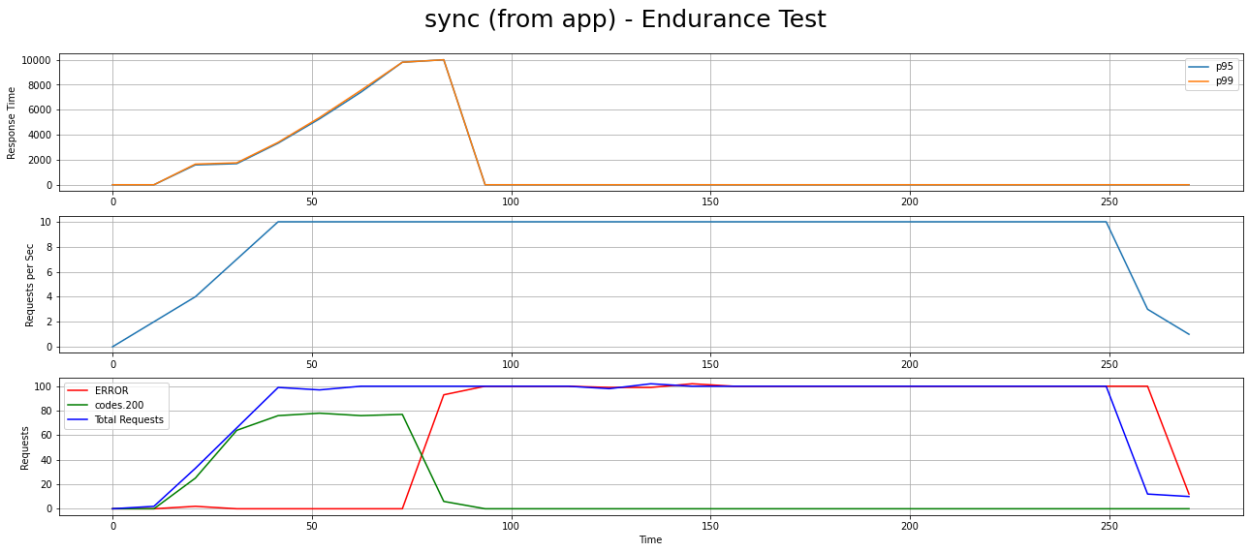
Spike test - Single node



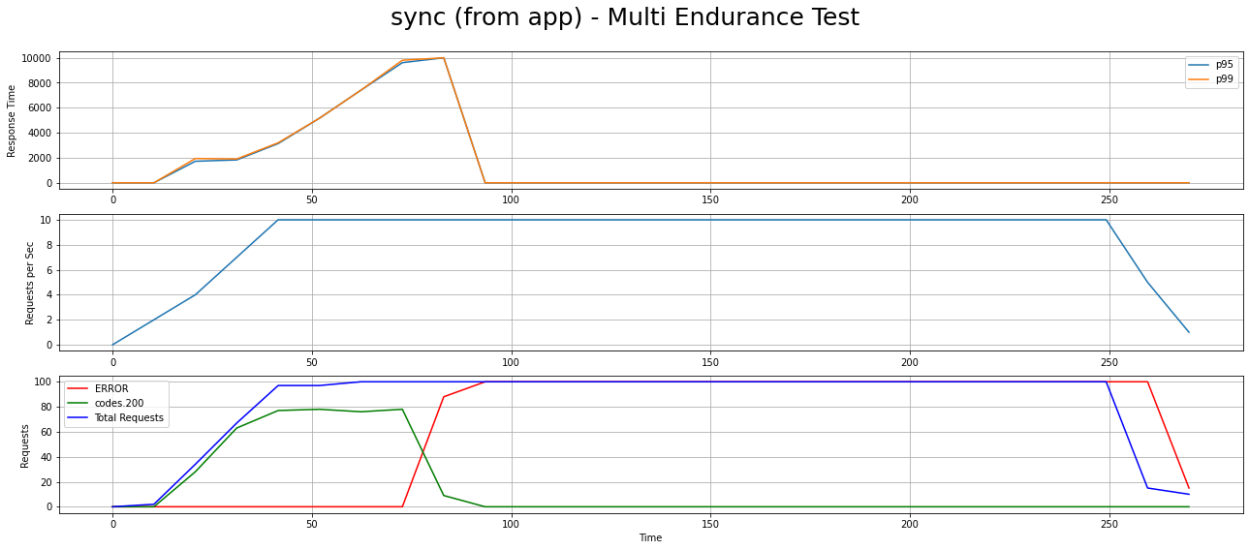
Spike test - Multi node



Endurance test - Single node



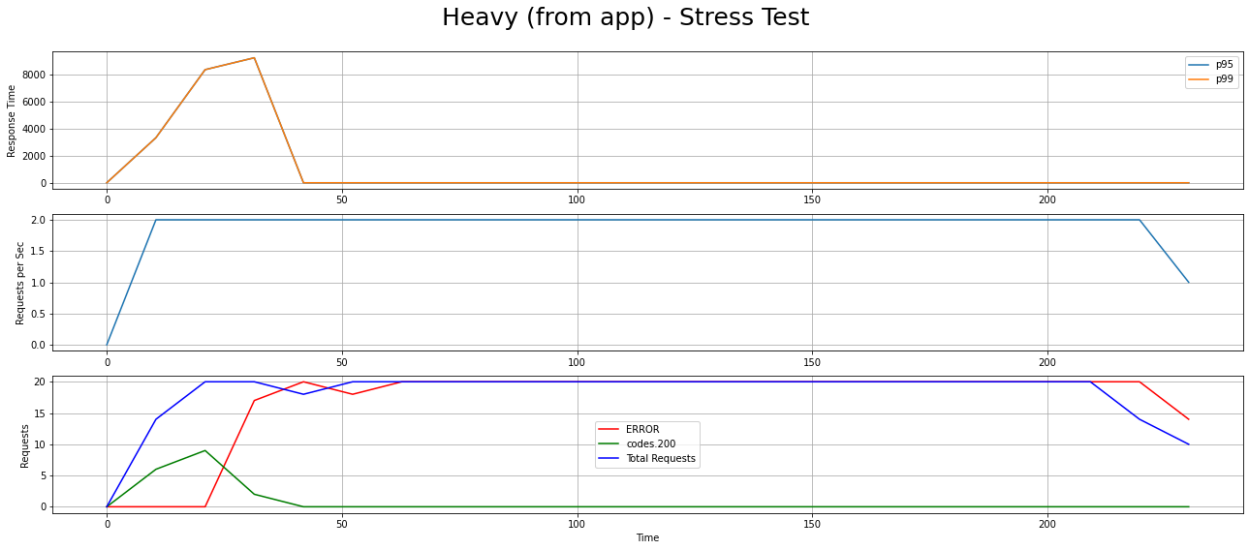
Endurance test - Multi node



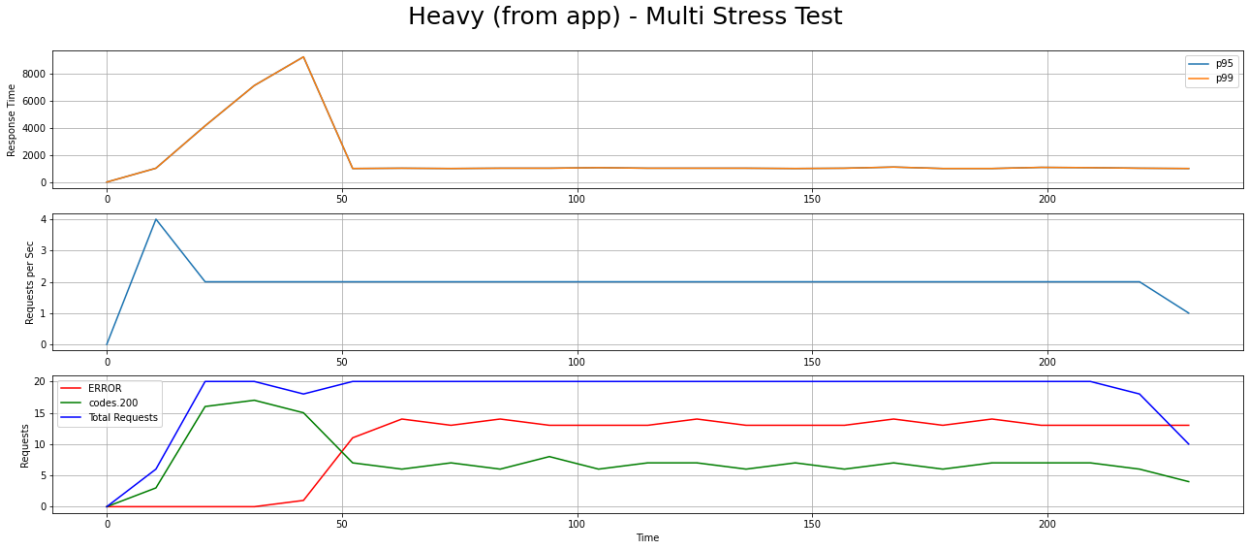
La diferencia entre multi o single node es minima, la aplicacion responde de una manera muy similar en ambas situaciones.

HEAVY

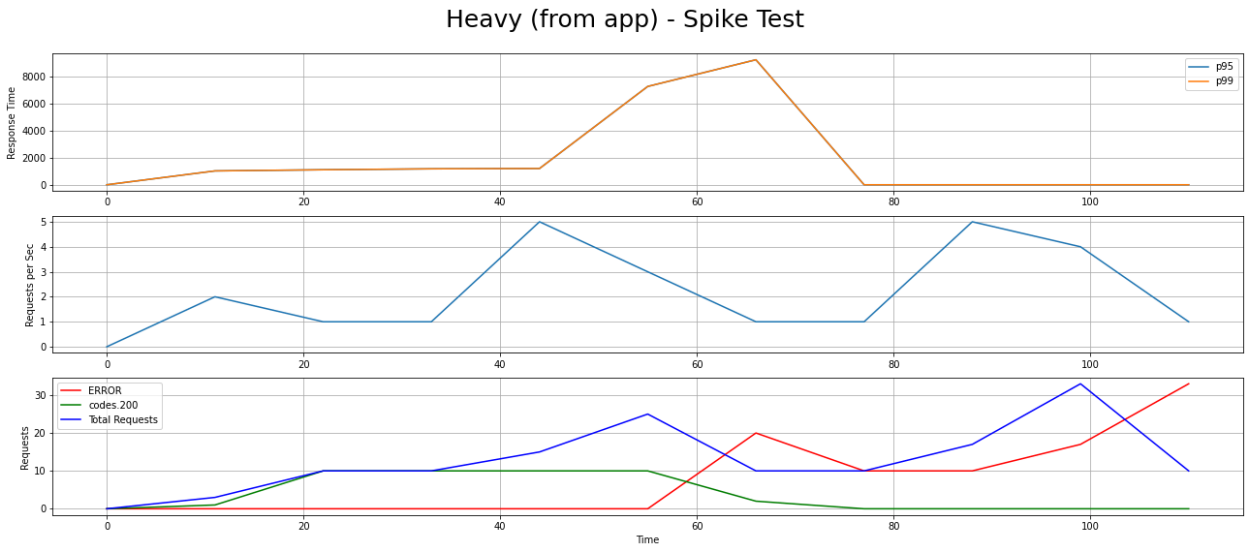
Stress test - Single node



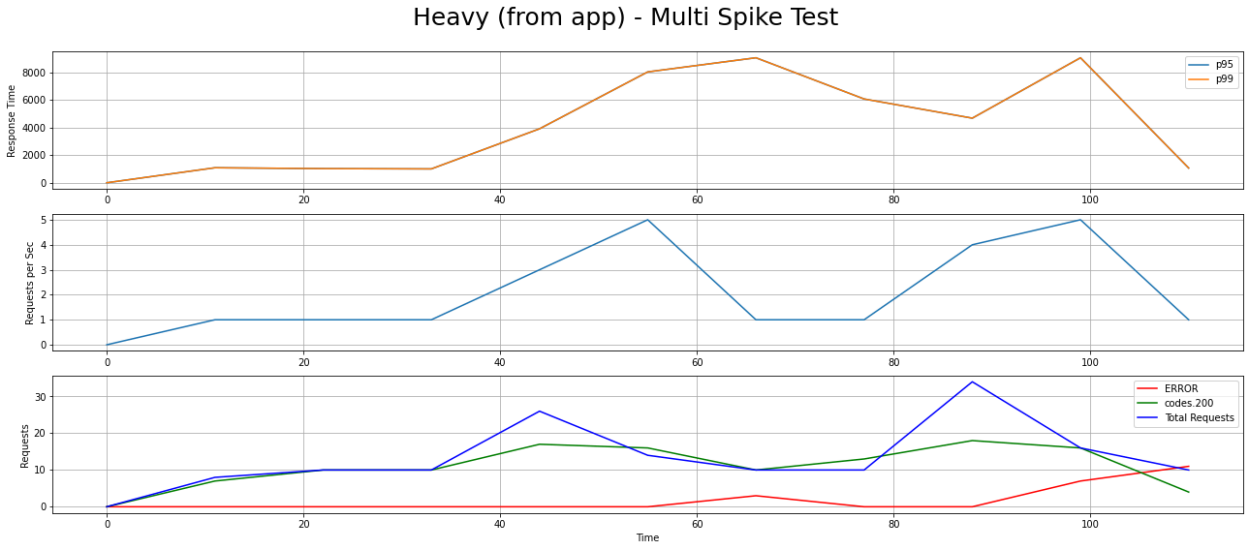
Stress test - Multi node



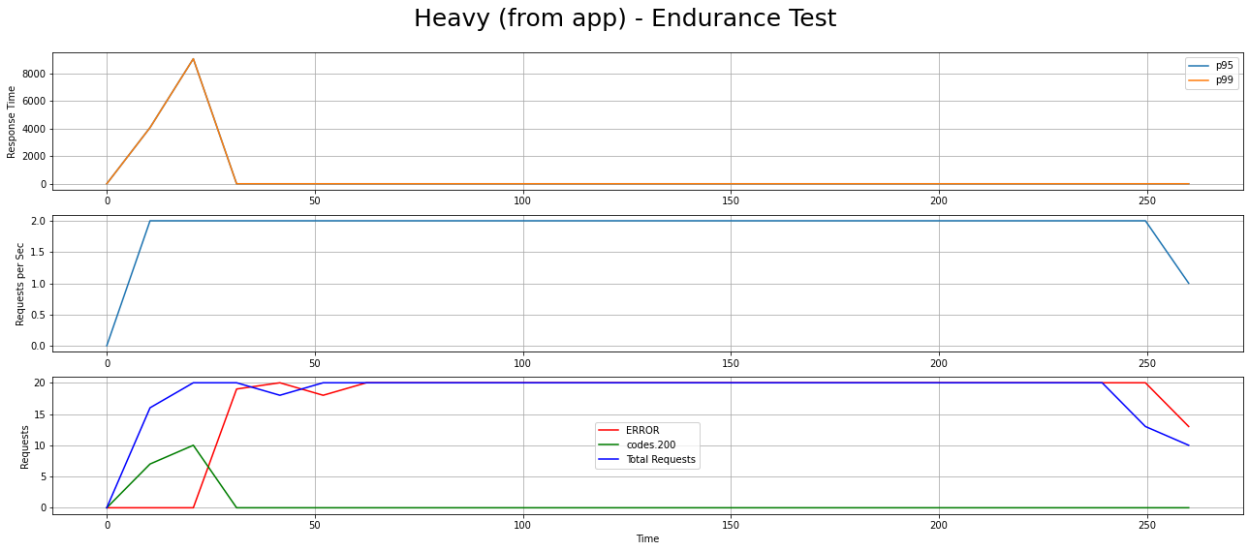
Spike test - Single node



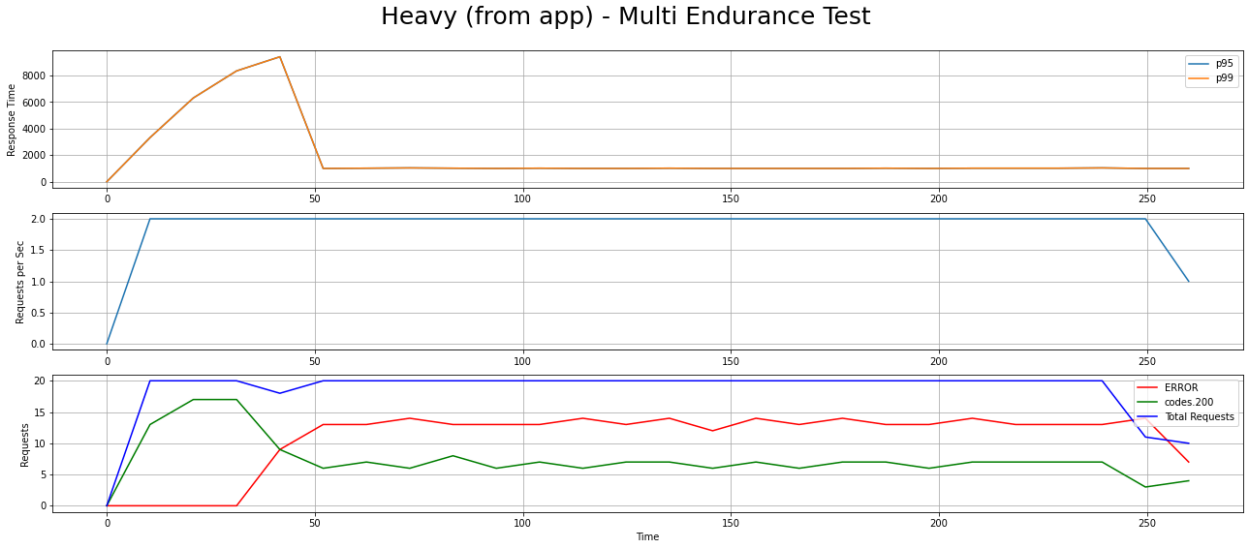
Spike test - Multi node



Endurance test - Single node



Endurance test - Multi node



Desde la primera entrega realizamos un cambio al test *Endurance* haciendo que la fase constante sea de 2 rps, ya que era lógico que la app responda bien para 1 rps.
Aquí el multi node saca provecho y logra marcar la diferencia en los tres tests. Sin embargo, el endurance test tiene muchas fallas, ya que es mejor que el single node (aguanta más tiempo funcionando) pero se “rompe” rápidamente de todas formas.

Sección 2

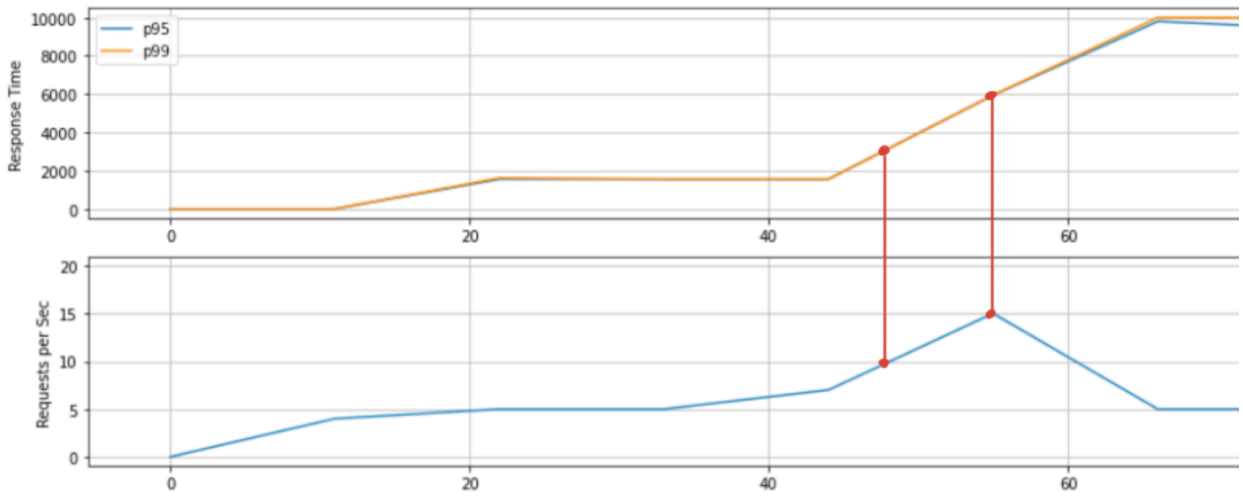
Análisis y caracterización

Sincrónico / Asincrónico

Luego de analizar ambos endpoints (9090 y 9091) vemos que el que tiene mejor response time, recibiendo más cantidad de request a la vez y a lo largo de los tests es el que recibe los requests en **9090**.
Asumimos que el servicio sincrónico es el de peor time response debido a que para sincronizarse debe bloquearse para cada request, por lo tanto tendrá una demora mayor.
Nuestra conclusión es que el servicio que escucha en el puerto 9090 es el servicio ***Asincrónico*** y el que escucha en el 9091 es el ***Sincrónico***.

Cantidad de workers (sincrónico)

Analizando los gráficos se SYNC (test SPIKE - Un nodo):



Observamos que mantiene el response time constante hasta que supera las 10RPS, que es cuando se dispara el response time. Asumimos que la cantidad de workers está entre 10-15.

Tiempo de Respuesta

El tiempo de respuesta de los endpoints puede analizarse a partir de los resultados de Artillery. Cuando el endpoint asincrónico de BBOX recibe una carga manejable, el tiempo de respuesta es de 1500 ms, mientras que el del sincrónico se acerca más a 1800-2000 ms, un poco más alto.

Sección 3

Sistema de Inscripción

Para esta sección, simularemos el comportamiento de inscripción a materias de una universidad. Para la simplificación de la prueba, implementamos los siguientes endpoints insertando un tiempo de espera o “response time” que debería tener en un escenario real de un sistema de inscripciones.
(para estas pruebas se tuvo que setear el timeout en 60s, ya que sino teníamos muchos timeouts, otras alternativas serían tener mayor cantidad de nodos o que los tiempos seteados en los endpoint sean menores)

Endpoint: /login

- Un usuario intenta loguearse al sistema (suponemos que ya está registrado en el mismo). Estimamos que tarda 100ms entre que verifica el nombre de usuario y verifica la contraseña.

Endpoint: /select_grade

- El usuario selecciona su carrera entre una lista de carreras disponibles. Suponemos 50ms como tiempo de respuesta.

Endpoint: /list_all

- El usuario quiere ver a que cursos se puede inscribir, tiempo estimado: 500ms

Endpoint: /list_courses

- El usuario quiere listar los cursos a los que se escribió, estimamos una espera de 300ms

Endpoint: /enroll

- El usuario se Inscribe a un curso. tiempo estimado: 100ms

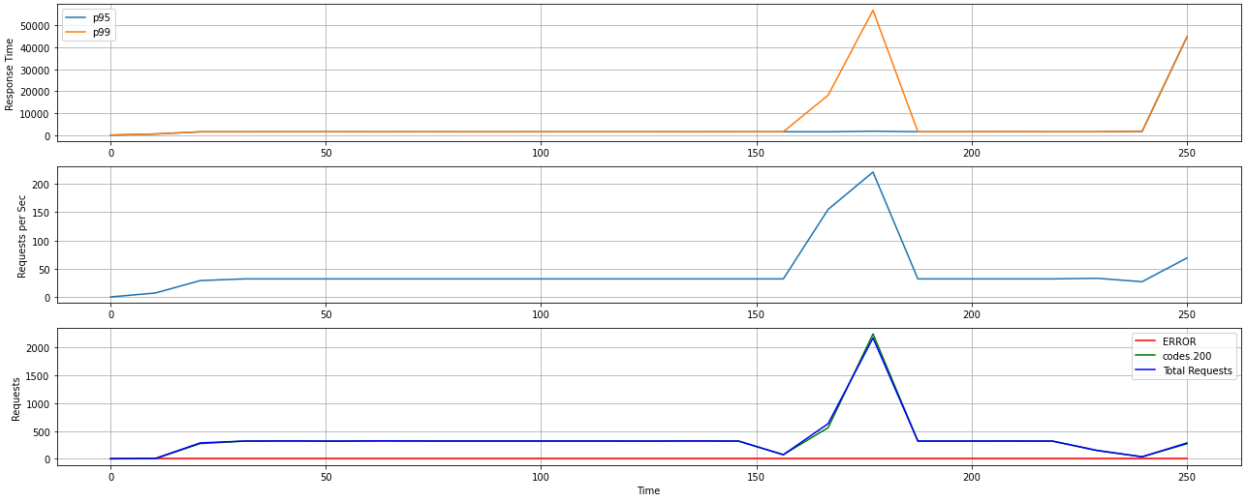
Endpoint: /logout

- El usuario cierra sesión, suponiendo que se debe guardar ciertos estados de este, estimamos una espera de 50ms

Caso 1:

Este escenario fue utilizado para poner a prueba el sistema y analizar su comportamiento, antes de realizar una prueba más compleja (y real).

Escenario: Entran 4 usuarios por segundo realizando las siguientes etapas -> Iniciar sesión, Seleccionar Carrera, Listar materias disponibles para inscribirse, Inscribirse a tres materias, Listar las materias a las que está inscripto y por último cerrar sesión. Este escenario se realizó durante 5 minutos, obteniendo:



Se puede observar que el sistema responde bien ante una carga no muy elevada, produciéndose un pico en el tiempo de respuesta al momento de tener que atender muchas request.

Este caso sirvió para entender que si bien el sistema no tiene una resistencia tan elevada, el caso usado no refleja la realidad, ya que todos los usuarios están realizando la misma tarea, la cual es bastante pesada. Por eso creamos un nuevo caso con más escenarios, pero también más usuarios creados por segundo.

Caso 2:

En este caso vamos a generar 10 usuarios por segundo, y también se van a atender diferentes request durante 5 minutos. Pero en este caso utilizaremos los siguientes escenarios, donde cada uno tuvo la misma probabilidad de realizarse por un usuario:

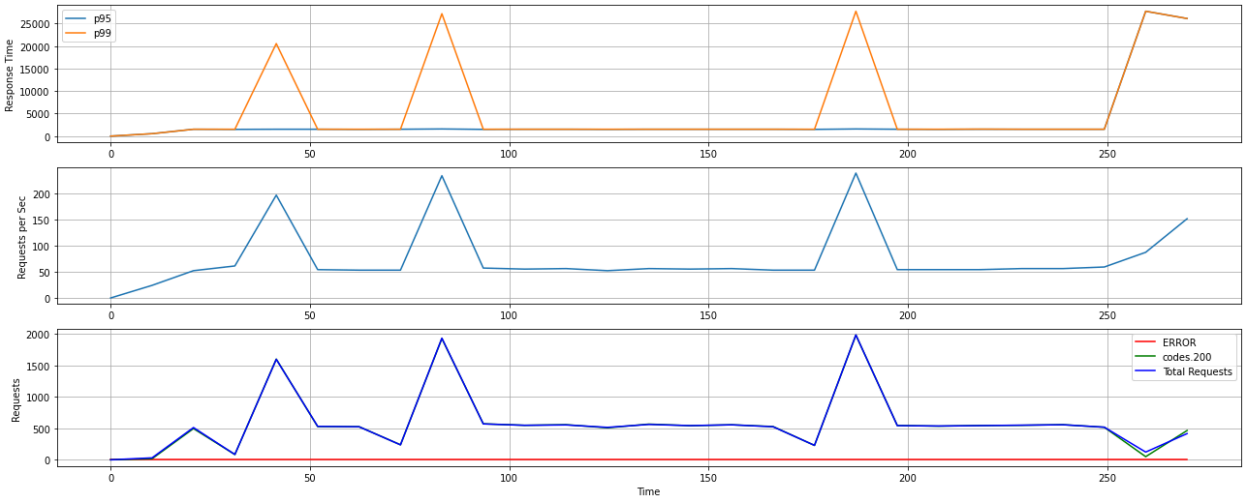
Escenario 1: Mismo que el usado por el caso anterior.

Escenario 2: Iniciar Sesión, Seleccionar Carrera, Listar materias inscriptas, Cerrar sesión.

Escenario 3: Iniciar Sesión, Seleccionar Carrera, Listar cursos disponibles, Cerrar sesión.

Escenario 4: Mismo que el caso anterior, pero anotandose solo en una materia.

El resultado obtenido fue:



Observamos que el tiempo de respuesta varió más, sin embargo los tiempos fueron lógicos y el pico máximo fue menor que en el caso anterior. Vemos que el sistema resistió bien un caso semejante a la realidad, logrando atender las peticiones de 3000 usuarios en 5 minutos.